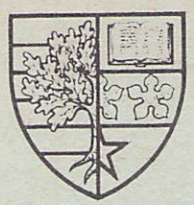
HERIOT-WATT UNIVERSITY



DEPARTMENT OF COMPUTER SCIENCE

ALGOL PROGRAMMING NOTES

A. BALFOUR
C. J. CROOK
Revised 25-6-69

HERIOT-WATT UNIVERSITY, EDINBURGH

DEPARTMENT OF COMPUTER SCIENCE

A L G O L

P R O G R A M M I N G

N O T E S

A. BALFOUR & C.J. CROOK

REVISED 25/6/69

LIST OF CONTENTS

1.0 INTRODUCTION 1

2.0 CONSTANTS 1

2.1 Integer Constants 1

2.2 Real Constants 2

2.3 Boolean Constants 4

3.0 VARIABLE NAMES, IDENTIFIERS, TYPE DECLARATIONS 4

3.1 Type Declarations 6

4.0 SIMPLE INPUT AND OUTPUT STATEMENTS 7

4.1 Simple Input (Elliott 4100 Algol) 8

4.2 Simple Output (Elliott 4100 Algol) 9

4.3 The Output of Messages (Elliott 4100 Algol) 10

5.0 ARITHMETIC ASSIGNMENT STATEMENTS 10

5.1 Simultaneous Assignment Statements 14

5.2 Standard Functions 14

6.0 STATEMENT BRACKETS, COMPOUND STATEMENTS, BLOCKS 16

7.0 EXAMPLES OF SIMPLE ALGOL PROGRAMS 17

8.0 LABELS, GOTO-STATEMENTS, CONDITIONAL STATEMENTS 19

8.1 The Goto-statement 21

8.2 Conditional Statements 22

8.2.1 The If-statement 22

8.2.2 The If-else-statement 27

8.2.3 The Nested Conditional Statements 29

8.2.4 Conditional Arithmetic Expressions 29

8.2.5 Conditional Designational Expressions 30

9.0 FOR-STATEMENTS 30

9.1 Value of Controlled Variable on Exit from a For-statement 37

10.0 ARRAYS AND SUBSCRIPTED VARIABLES 38

11.0 SWITCHES 46

12.0 BLOCK STRUCTURE - THE SCOPES OF QUANTITIES 51

13.0 COMMENTS 56

14.0 BOOLEAN VARIABLES AND EXPRESSIONS 57

14.1 Logical Operators 58

15.0 OWN VARIABLES 63

16.0 PROCEDURES 64

16.1 Ordinary Procedures 64

16.2 Procedures with Formal Parameters 69

16.3 The Value List - Execution of a Procedure Statement .. 76

16.4 Function Procedures 83

16.5 Recursive Procedures 88

16.6 Side Effects 90

APPENDIX 1 Algol 60 Basic Symbols 92

APPENDIX 2 List of References 94

APPENDIX 3 Flow Chart Practice 95

1.0 Introduction

A modern computer will accept a variety of programming languages, typically

(i) A basic "machine code" in which it is difficult to write programs, since a machine code is designed with a particular computer in mind and not the unfortunate individuals using that computer.

(ii) An assembly language which uses convenient mnemonics in place of machine code instructions and hence is more suitable for the human user.

(iii) "High level" languages such as ALGOL 60 and FORTRAN (for scientific work) and COBOL (for commercial work). These are "machine - independent" languages which have been designed to suit human users and not specific computers.

The aim of this document is to acquaint the reader with the high-level programming language Algol 60. This was devised by an international committee in 1960 and modified two years later to produce the language as we know it today. The language is precisely defined in the document

Revised Report on the Algorithmic Language Algol 60

which was published in January, 1963. The reader is warned that this is not an easy document to read and is better left alone until the basic essentials of the language are fairly well understood.

2.0 Constants

An Algol program may contain constants of three different types, namely integer constants, real and Boolean constants. We consider each in turn.

2.1 Integer Constants

An integer constant is merely an ordinary number written without a decimal point. An integer constant may be positive, negative or zero. If it is unsigned, it is assumed to be positive.

Example 1 The following are acceptable integer constants:-

```
 373
-26
  0
+1234
 -0
```

Example 2 The following are not acceptable integer constants:-

```
  0.0     (decimal point is not permitted)
3,678     (embedded comma is not permitted)
```

One final point with regard to integer constants - there is almost always a size restriction on integer constants and this varies from computer to computer. For example in Elliott 4100 Algol all integer constants must lie between

-8388608 (i.e. -2^{23}) and 8388607 (i.e. $2^{23}-1$)

2.2 Real Constants

A real (or decimal) constant in Algol can be written in a number of different ways, namely

(i) as an unsigned integer

(ii) as a sequence of digits with a decimal point before one of them.

(iii) as a subscript ten, $_{10}$, followed by an integer. This represents a power of ten.

(iv) as either of the forms (i) and (ii) followed by the form (iii) as an exponent part.

(v) as one of the forms (i), (ii), (iii) and (iv) above preceded by a sign + or - .

A number of examples of real constants will indicate what is permissible in Algol 60.

Example 1 The following are acceptable real constants

3615

0

12.0

.61329

$_{10}13$

$_{10}-6$

$2.357_{10}-3$

$.615_{10}8$

-3124

-9.8

$-_{10}6$

$-1.786_{10}8$

$-0.22_{10}5$

Example 2 The following are not acceptable as real constants:-

678. (decimal point must precede a digit)

$-5._{10}8$ (decimal point must precede a digit)

$3.2_{10}1.7$ (exponent part must not contain a decimal point)

$31.6\times_{10}-6$ (multiplication sign is not permitted)

As with integer constants, there is almost always a restriction on the magnitude of a real constant and this depends on the computer system being used. In Elliott 4100 Algol all real constants must lie between the (approximate) limits

$$-5.8 \times 10^{76} \text{ and } 5.8 \times 10^{76}$$

and any real constant which is less than 4.3×10^{-76} in magnitude cannot be distinguished from zero.

Note also that, whereas integer constants are held within the store of a computer with complete accuracy, real constants are held to a certain precision which, once again, depends on the computer system being used. In Elliott 4100 Algol, for example, real constants are held to an accuracy of about 11 significant decimal digits.

2.3 Boolean Constants

There are two Boolean or logical constants, namely, true and false. Note that true and false are two of the so called Algol basic symbols and, as such, are always underlined when written.

3.0 Variable Names, Identifiers, Type Declarations

The names used for the variables in a program, and for other purposes such as labelling parts of a program, are known as identifiers. These are created by the programmer according to the fundamental rule:-

An identifier is any combination of letters

A B C D - - - - - - - - - - - - - - - - X Y Z
a b c d - - - - - - - - - - - - - - - - x y z

and digits

0 1 2 3 4 5 6 7 8 9

which begins with a letter.

Example 1 The following are acceptable identifiers:-

X	pressure
y	time
alpha	volume
VW1600	HSC369E
H2SO4	willy
x23	trouble

Example 2 The following are not acceptable identifiers

BBC-2	(hyphen is not allowed)
76 trombones	(does not begin with a letter)
Riley 3.5	(decimal point is not allowed)
AB"	(priming" is not allowed)
π	(Greek letters are not allowed)

Note that

(i) It is good practice in programming to create meaningful identifiers. Thus, for example, in a calculation involving a pressure, a temperature and a volume, it would be much more sensible to create the three identifiers

PRES
TEMP
VOL

than to create the three identifiers

CLARENCE
ORANGE
WOMBAT

(ii) In Elliott 4100 Algol only the first six characters of an identifier are of significance to the system. Thus, for example, the identifiers CONDENSER and CONDENSATION are regarded as identical.

(iii) Although both upper and lower case letters are permissible, it should be kept in mind that upper case letters only are available on most data preparation devices. For example the IBM 029 card punches which can be used for the preparation of programs written in Elliott 4100 Algol do not possess lower case letters. It is therefore advisable to avoid using identifiers which are identical apart from case differences e.g. x and X.

3.1 Type Declarations

Since variables in an Algol program can assume values either of type integer, real or Boolean (c.f. constants) it is necessary to indicate to the computer system whether a particular identifier represents a variable which will assume integer values or a variable which will assume real values or a variable which will assume Boolean values. This is done by means of type declarations.

Example 1 In a program we might write

```
real      x, y, willy;
integer   i, j, pip, count;
Boolean   test
```

This indicates that

(i) the identifiers x,y,willy represent variables which will assume real values later in the program.

(ii) the identifiers i,j,pip,count represent variables which will assume integer values later in the program.

(iii) the identifier test represents a variable which will assume logical values (true and false) later in the program.

Note that

(i) a type declaration consists of (at the moment) one of the basic Algol symbols real, integer or Boolean followed by a list of identifiers separated by commas.

(ii) a semi-colon is used to separate one type declaration from another.

(iii) the spacing of the text in an Algol program is immaterial and is ignored at all times by the computer system. For example, the above declarations could equally well have been written

real x,y,willy; integer i,j,pip,count; Boolean test

(iv) in Elliott 4100 Algol the order in which type declarations are made is unimportant.

Example 2

```
integer pip,squeak,willy;
real  x,y1,y2,coeff
```

4.0 Simple Input and Output Statements

The 1963 Revised Report on the Algorithmic Language Algol 60 does not specify any structure for input and output statements. Because of this, implementations of Algol 60 vary considerably with regard to the input and output of quantities. We consider here simple input and output statements for Elliott 4100 Algol. For more detailed information the reader should consult the Computer Department document

A Plain Man's Guide to Elliott 4100 Algol.

4.1 Simple Input (Elliott 4100 Algol)

A simple input statement in Elliott 4100 Algol has the form

read <list>

where <list> is a list of identifiers separated by commas. Its effect is to cause numbers to be read in from cards and to be assigned to the variables whose identifiers are given in the list.

Example 1 The statement

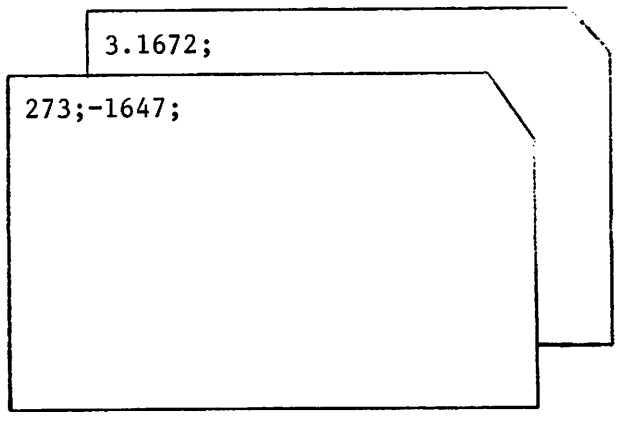
```
read    x,y,pip,count
```

means "read in four numbers from cards and assign their values to the variables x,y,pip and count".

Example 2 An Algol program might contain the following

```
integer     i,j,count,m,n;
real        x,y,w,result;
read        m,n,x
```

and the read statement would cause three numbers to be read in from cards and assigned to the variables m,n and x. For example, if the cards read were



then m would be given the value 273, n the value -1647 and x the value 3.1672.

Note how numbers to be input can be separated out by semi-colons.

4.2 Simple Output (Elliott 4100 Algol)

A simple output statement in Elliott 4100 Algol has the form

print <list>

where <list> is a list of identifiers separated by commas. Its effect is to cause the values of the variables whose identifiers are given in the list to be printed out via the lineprinter. Each value is printed at the start of a new line.

Example 1 The statement

print i,j,willy,x,temp

means "print out via the lineprinter the values of the variables i, j, willy, x and temp".

Example 2 If i,j are of type integer and x,y,q of type real then the statement

print i,j,x,y,q

might produce the output

```
- 3 6 1 5 2
        3 4
  3 6 5 . 7 1 4 2 6
- 3 . 9 1 5 6 3 2 8
- 4 9 3 . 2 1 5 8 7
```

in which each value output has been printed at the start of a new line. If it were required to print the values of i,j,x,y and q on the same line then we would write

print i,SAMELINE,j,x,y,q

and the prefix setting procedure (don't worry about this!) SAMELINE would cause the values of j,x,y and q to be printed on the same line as the value of i.

4.3 The Output of Messages (Elliott 4100 Algol)

A message can be printed out in Elliott 4100 Algol by enclosing it in the single string quotation marks ´ and `, and inserting it in the list of a print statement. Note that messages are not automatically output at the start of a new line.

Example 1 print ´standard deviation = `

Example 2 print x,´feet`,y,´inches`

Example 3 print ´values of real and imaginary parts are`,x1,x2

Example 4 print xbar,SAMELINE,sigma,´are the required values`

5.0 Arithmetic Assignment Statements

The arithmetic assignment statement is the Algol way of evaluating some expression to obtain a new value for some variable. Each assignment statement has the form

<identifier> := <arithmetic expression>

and indicates that the variable whose identifier is on the left of the special symbol := (read as "becomes" or "is given the value of") is to be given the value of the arithmetic expression written on the right of the := sign.

Example 1 y:= (a+b)×(3×c-1.69)

means that the variable y is to be given the value of the expression (a+b)(3c-1.69). This does not make sense, of course, unless a,b,c have already been given values e.g. by means of a read-statement.

Example 2 Result:= pip-3.6×pop+5.7×pressure

means that the variable Result is to be given the value of the expression pip-3.6pop+5.7pressure.

Let us now consider arithmetic expressions in more detail. An arithmetic expression may contain constants, variables and other expressions linked by the following arithmetic operators

arithmetic operator	meaning
+	addition
-	subtract
× *	multiplication
/	division
÷ "DIV"	integer division (see note (iv) below)
↑	exponentiation (a↑b≡a^b)

and rounded brackets (and) may be freely inserted where necessary or where it improves readability to do so. In evaluating an arithmetic expression the following order of precedence is used

(i) anything in rounded brackets

(ii) ↑

(iii) ×, /, ÷

(iv) +, -

(iii) variables of type real and integer may be mixed freely in arithmetic assignment statements, since automatic type conversion will be carried out where necessary. The result of an operation is always of the same type (real or integer) as the operands, except when operands are mixed, in which cases a result of type real is always produced.

(iv) a/b is always of type real, even when a,b are of type integer. If a and b are both integers there is a special division operator ÷ which is such that a÷b produces a result of type integer equal to the value of a/b truncated. For example 7÷4 = 1, 33÷6 = 5, (-26)÷10 = -2 etc..

(v) in Elliott 4100 Algol a↑b is always of type real, even when a,b are of type integer.

(vi) the result of certain operations is undefined e.g.

$$a\uparrow b \quad \text{with} \quad \begin{cases} a=0 \text{ and } b\leq 0 \\ a<0 \text{ and any real } b \end{cases}$$

$$a/b \quad \text{with} \quad b=0$$

Example 4 The following assignment statements are impermissible:-

(a) PRES := aX-b

(b) temp := 3.14 DEG

(c) x+y := 2↑-1.272

(d) count:= I÷(2↑4)

5.1 Simultaneous Assignment Statements

When the same value is to be assigned to several variables Algol permits the use of a simultaneous assignment statement of the form

<identifier> := <identifier> := ---------- <identifier> := <arithmetic expression>

Example 1 The simultaneous assignment statement

```
x := y := pip := temp := (a+b)↑4
```

assigns the value of (a+b)↑4 to the variables x,y,pip and temp.

Example 2 The simultaneous assignment statement

```
p := q := alpha := n×(n+1)
```

assigns the value of n×(n+1) to the variables p,q and alpha i.e. it is equivalent to the statements

```
p := n×(n+1);
q := n×(n+1);
alpha := n×(n+1)
```

One further point about simultaneous assignment statements - the identifiers listed in a simultaneous assignment statement must all be of the same type. Thus in Example 1 above, x,y,pip and temp must either be all of type real or all of type integer.

5.2 Standard Functions

The following standard functions are available in all implementations of Algol 60. The first group produces results of type real, the second results of type integer, where X is any arithmetic expression.

Function	Value
sin(X)	sine of X radians
cos(X)	cosine of X radians
arctan(X)	principal value in radians of $\tan^{-1} X$
sqrt(X)	$+\sqrt{X}$
exp(X)	e^X
ln(X)	$\log_e(X)$
abs(X)	$\lvert X \rvert$
sign(X)	-1 if X<0, 0 if X=0, +1 if X>0
entier(X)	greatest integer less than or equal to X

As the following examples indicate, these standard functions may be used freely within arithmetic expressions.

Example 1 y:= (exp(a+b)+3.0)×(cos(t)-t↑4)

Example 2 x:= (-b+sqrt(b×b-4×a×c))/(2×a)

Example 3 c:= sqrt(a×a+b×b-2×a×b×cos(3.14159×ANGC/180))

Example 4 PI:= 4.0×arctan(1.0)

Example 5 q:= sin(cos(u)) - cos(sin(u))

Note also that in Elliott 4100 Algol a number of other standard functions are available. These include

Function	Value
tan(X)	tangent of X radians
arcsin(X)	principal value, in radians, of $\sin^{-1} X$
arccos(X)	principal value, in radians, of $\cos^{-1} X$

6.0 Statement Brackets, Compound Statements, Blocks

It is often desirable to make a number of statements act as a single entity for reasons which will become apparent later. If this is to be done the group of statements must be preceded by the word begin and followed by the word end. The group is then called a compound statement. The Algol basic symbols begin and end are called statement brackets since they serve to denote the limits of a group of statements in much the same way as ordinary brackets specify the limits of a mathematical expression which is to be treated as a unit in later operations.

Example 1 The following is a compound statement

```
begin
     read a,b;
     y:= a↑3+b↑3 - 3×(a+b)↑2;
     print y
end
```

A block has the appearance of a compound statement, being enclosed between begin and end, but in addition it includes type declarations immediately following the begin. In other words a block has the form

```
begin
     <declaration>;
     <declaration>;
           ¦
           ¦
           ¦
     <declaration>;
     <statement>  ;
     <statement>  ;
           ¦
           ¦
           ¦
     <statement>
end
```

Example 2 The following is a block in Algol

```
begin
     integer i,j;
     real    alpha,x,y;
     read    alpha,j;
     i:= j+6;
     x:= cos(alpha) - 1.65;
     y:= (alpha - 3.7)↑4.6 + exp(x)
end
```

Every Algol program must consist of at least one block i.e. it must start with begin, be followed by the declarations and the statements of the program, and terminate with end. We are now in a position to consider examples of complete Algol programs.

7.0 Examples of Simple Algol Programs

Example 1 Given values for a,b on cards write a program to evaluate and print x,y where

$$x = \left(\frac{a+b}{a-b}\right)^{1.65} \quad , \quad y = \left(\frac{a-b}{a+b}\right)^{3.76}$$

```
begin
      real a,b,x,y;
      read a,b;
      x:=  ((a+b)/(a-b))↑1.65;
      y:=  ((a-b)/(a+b))↑3.76;
      print x,y
end
```

Note the use of the semi-colon as a "separator". This is the result of one of the Algol rules for punctuation:-

> Every declaration and statement must be followed by either a semi-colon, the basic symbol end or the basic symbol else.

Note also that every variable used in a program must be declared.

Example 2 Given on cards are values for the sides a,b and the angle C of ΔABC. C is given in degrees. Write a program to calculate the length of the remaining side using the formula

$$c^2 = a^2+b^2 - 2abcosC$$

```
begin
     real a,b,c,ANGC;
     read a,b,ANGC;
     c:= sqrt(a×a+b×b-2×a×b×cos(3.141593×ANGC/180));
     print c
end
```

Example 3 Given on cards are three numbers x, y and z corresponding to the sum of money £x:y:z. Write a program to convert this sum to pence.

```
begin
     integer x,y,z, numpence;
     read    x,y,z;
     numpence:= 240×x+12×y+z;
     print   numpence
end
```

Example 4 Given on cards are the values of the coefficients a,b,c of the quadratic equation $ax^2+bx+c=0$. Write a program to evaluate and print its roots using the formula

$$x1,x2 = \frac{-b\pm\sqrt{b^2-4ac}}{2a}$$

You may assume that the roots are real and that $a\neq 0$.

```
begin
     real  a,b,c,t,x1,x2;
     read  a,b,c;
     t:= sqrt(b×b-4×a×c);
     x1:= (-b+t)/(2×a);
     x2:= (-b-t)/(2×a);
     print x1,SAMELINE,x2
end
```

A better version, which avoids the loss of significant figures when one of the roots is very small, is:-

```
begin
     real  a,b,c,x1,x2;
     read  a,b,c;
     x1:= (-b-sign(b)×sqrt(b×b-4×a×c))/(2×a);
     x2:= c/(a×x1);
     print x1,SAMELINE,x2
end
```

8.0 Labels, Goto-statements, Conditional Statements

Normally in an Algol program the statements are executed in the order in which they are written. Thus, for example, if a program contains the statements

```
i:= i+1;
y:= a×x↑2+b×x+c;
z:= exp(t+0.5)
```

then the computer will first obey the statement

```
i:= i+1
```

followed by the statement

```
y:= a×x↑2+b×x+c
```

and finally the statement

```
z:= exp(t+0.5)
```

Quite often, however, we may wish to break this natural order and either repeat some statements, which have already been obeyed, or omit some statements and go on to a later part of the program.

In order to be able to do this we must have some means whereby we can refer to particular statements. In Algol this is achieved by the use of labels which we can attach to statements.

According to the Revised Report a label is either a positive integer or an identifier (which is not being used for some other purpose in the same block) but since very few implementations of Algol accept positive integers as labels we shall adhere to the alternative method of construction. To attach a label to a statement is simple - write the label before the statement and separate them from each other by a colon.

Example 1 Consider the following statements

```
      p:= 2×a+17;
   L1: x:= -b+sqrt(p);
again: alpha:= alpha×alpha-sqrt(13.6);
      y:= y-7;
  out: print x,y, alpha
```

Then the statement

```
x:= -b+sqrt(p)
```

has been given the label L1, the statement

```
alpha:= alpha×alpha - sqrt(13.6)
```

has been given the label again and the statement

```
print x,y,alpha
```

has been given the label out.

Three further points with regard to labels:-

(a) Two or more statements in the one block must not be given the same label.

(b) A labelled statement may itself be given a label e.g.

```
L1:out:alas:y:=a×(r↑n-1)/(r-1)
```

(c) The basic symbols begin and end may also be labelled, e.g.

```
      y:= (x+1)↑3.6 - t↑4;
      print y ;
AA:end
```

[In actual fact the label is not attached to the basic symbol but to a "dummy" statement immediately preceding the basic symbol!!]

8.1 The Goto-statement

The normal sequencing in a program can be broken at any time by using a goto-statement which has the form

goto <label>

where <label> is the name of an actual label in the program.

Example 1 The statement

goto L3

means "transfer control to the statement labelled L3 i.e. obey next the statement labelled L3".

Example 2 Values for a_1, b_1, c_1, a_2, b_2, c_2, a_3, b_3, c_3, - - - - - - - - are given on a deck of cards. Write a program to evaluate and print $(a_i^2+b_i^2)/c_i$ for i = 1, 2, 3, - - - - - - - - .

```
begin

        real  a,b,c,y;
again:  read  a,b,c;
        y:=   (a×a+b×b)/c;
        print y;
        goto  again

end
```

Note that a rather inelegant "halt" would occur when the data cards were exhausted. How this can be avoided will be indicated shortly.

8.2 Conditional Statements

The goto-statement is sometimes referred to as an unconditional jump statement since control is transferred unconditionally each time the statement is obeyed. Sometimes however we may only wish to transfer control if a particular criterion or condition is satisfied. Conditional statements enable us to achieve this (and other things).

8.2.1 The If-statement

The Algol if-statement has the form

if <Boolean expression> then <statement>

where

(i) <Boolean expression> is any logical expression i.e. an expression which has either the value "true" or "false".

(ii) <statement> is any Algol statement which is not itself conditional i.e. do not follow then immediately by if

and is a means for deciding whether or not the statement following the basic symbol then is to be obeyed. If the Boolean expression has the value "true" then the statement is obeyed; if it has the value "false" then it is not.

Example 1 The Algol statement

```
if x+y>z then p:= (q+r)↑5
```

means "if the value of x+y is greater than the value of z then p is given the value of $(q+r)^5$ and control is transferred to the next statement in the program.

On the other hand if the value of x+y is not greater than z ignore the statement p:=(q+r)↑5 and transfer control to the next statement in the program".

Example 2 The Algol statement

```
if m = n+2 then goto again
```

means "if the value of m is equal to the value of n+2 then transfer control to the statement labelled again. On the other hand if the value of m is not equal to the value of n+2 then ignore the statement goto again and transfer control to the next statement in the program".

Example 3

```
if x+y < (p+7.6)↑5 then y:= (p+q)×cos(t)
```

Example 4

```
if j≠17 then begin
                read x;
                S1:= S1+x;
                S2:= S2+x×x
             end
```

Note that the Boolean expressions which are normally used in an if-statement have the form

<arithmetic expression><relational operator><arithmetic expression>

where <relational operator> is one of the symbols

$$< \ , \ \leq \ , \ = \ , \ > \ , \ \geq \ , \ \neq$$

One word of warning - it is inadvisable to compare the values of two real variables by means of the relational operator = . This is because the values of real variables may not be held with complete accuracy inside a computer and hence values which are theoretically equal may differ slightly in internal representations.

Example 5 If x,y are real variables then the statement

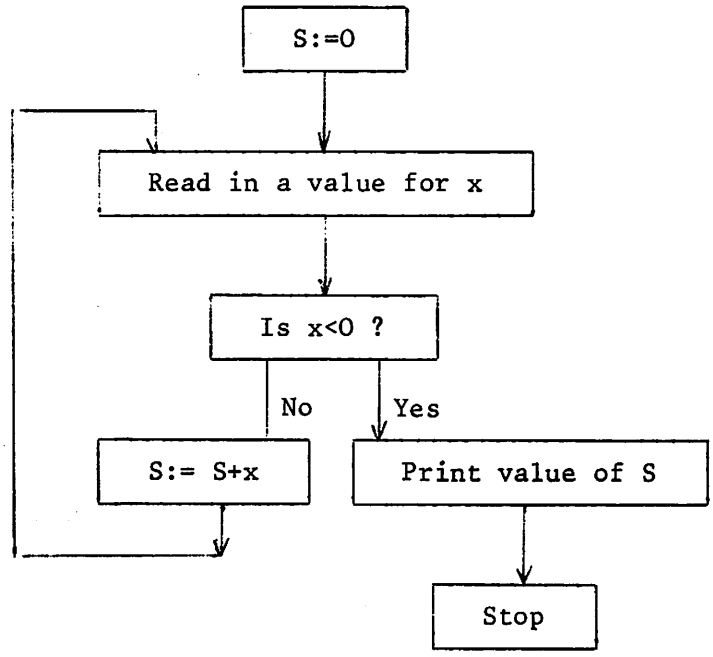
```
if  abs(x-y)<10-12 then goto L1
```

is very much better than the statement

```
if  x=y then goto L1
```

Example 6 Positive numbers x_1, x_2, x_3, - - - - - - - - , x_n are given on cards and terminated by -1. Write an Algol program to calculate and print the sum S of the positive numbers.

A possible flow-chart is



leading to the program

```
begin
        real  x,S;
        S:=0;
  again: read x;
        if x<0 then goto out;
        S:=S+x;
        goto again;
    out: print S
end
```

Note how a sum can be formed by working round a "loop" of instructions. Note also how it is essential to "initialise" S by setting it equal to zero before the loop is entered. We shall encounter these ideas and similar ones again and again.

Example 7 Given on cards is a list of candidate numbers, each candidate number being followed by the candidate's marks in mathematics, physics and chemistry. Write an Algol program to print out the candidate numbers of those candidates who scored more than 50 in mathematics, 45 in physics and 60 in chemistry. Once again the list of data is terminated by -1.

```
        begin
              integer cn,m,p,c;
           L1: read cn;
              if cn<0 then goto L2;
              read m,p,c;
              if m≤50 then goto L1;
              if p≤45 then goto L1;
              if c≤60 then goto L1;
              print cn;
              goto L1;
    L2: end
```

Example 8 Positive values for x_1, x_2, x_3, - - - - , x_n are given on cards and terminated by -1.0. For values of x greater than 2 only, print out the value of x followed by the corresponding value of $(x^4+x^2+1)/(x^3-8)$.

```
        begin
              real x,y;
         again:read x;
              if x<0 then goto out;
              if x>2 then begin
                                  y:= (x↑4+x×x+1)/(x↑3-8);
                                  print x,SAMELINE,y
                          end;
              goto again;
   out: end
```

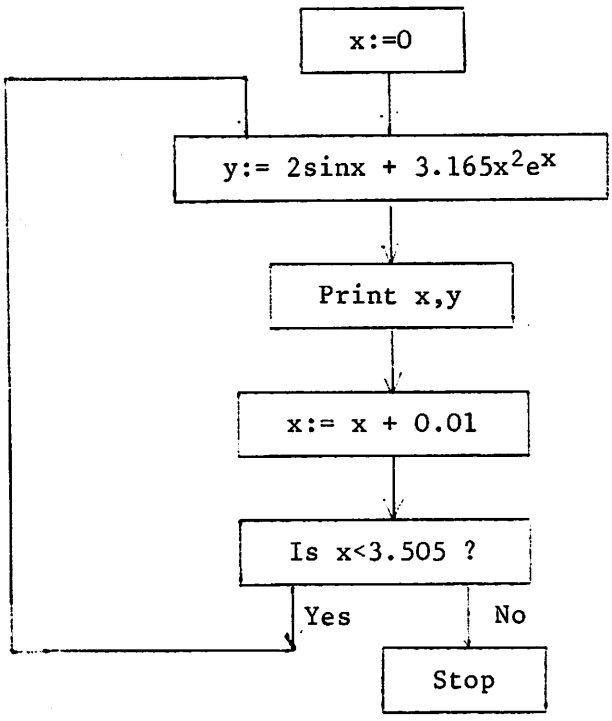
Note that in this example there is an if-statement in which the statement following the basic symbol then is a compound statement. Note also that this example illustrates one more Algol punctuation rule:-

> The basic symbol end must be followed by either semi-colon, the basic symbol end or the basic symbol else.

Example 9 Write an Algol program to evaluate and print the expression $y = 2\sin x + 3.165x^2e^x$ for x = 0(0.01)3.5.

N.B. [The notation x = 0(0.01)3.5 means x = 0,0.01, 0.02, 0.03, ----, 3.5]

A possible flow diagram is



leading to the program:-

```
begin
        real x,t;
        x:=0;
    L1: y:= 2×sin(x)+3.165×x×x×exp(x);
        print x,SAMELINE,y;
        x:= x+0.01;
        if x<3.505 then goto L1
end
```

Note that it is safer to write 3.505 than 3.5 (why?).

8.2.2 The If-else-statement

The if-statement is a means for deciding whether or not a statement is to be executed. The if-else-statement which has the form

if <Boolean expression> then <statement> else <statement>

is a means for deciding which of two alternative statements is to be executed. Note that

(i) the statement following the basic symbol then must neither be conditional nor a for-statement [See Section 9.0] i.e. do not follow then by if or for.

(ii) the statement following the basic symbol else may be any Algol statement.

Example 1 if x>y then p:=q+1 else goto alas

Example 2 if i≠j+1 then y:=y+17 else print u, pip

Example 3 Given on cards are values for an integer n and a real variable x. Write a program to evaluate and print

$x^2+2x(\cos x+e^x)-3$ if n=1
$x\sin x-x^2\cos x+3\ell nx$ if n=1

```
begin
      real x,y;
      integer n;
      read n,x;
      if n=1 then y:= x×x+2×x×(cos(x)+exp(x))-3
             else y:= x×sin(x)-x×x×cos(x)+3×ℓn(x);
      print x,SAMELINE,y
end
```

Example 4 Given on cards are values for a_1, b_1, c_1, a_2, b_2, c_2 the coefficients of the simultaneous equations $a_1x+b_1y = c_1$ and $a_2x+b_2y = c_2$. Write an Algol program to evaluate and print x,y.

We base our program on the formulae

$$x = \frac{c_1b_2 - c_2b_1}{a_1b_2 - a_2b_1} \quad , \quad y = \frac{a_1c_2 - a_2c_1}{a_1b_2 - a_2b_1} \quad .$$

```
begin
      real a1,b1,c1,a2,b2,c2,denom,x,y;
      read a1,b1,c1,a2,b2,c2;
      denom:= a1×b2 - a2×b1;
      if denom ≠ 0 then begin
                              x:= (c1×b2-c2×b1)/denom;
                              y:= (a1×c2-a2×c1)/denom;
                              print x,SAMELINE,y
                        end
                   else print 'youre joking'
end
```

Note carefully the punctuation of this program.

8.2.3 Nested Conditional Statements

It has already been pointed out that the statement following the basic symbol else in an if-else-statement may be any Algol statement. In particular it may be another conditional statement, so that statements of the form

if <BE1> then <S1> else if <BE2> then <S2>

and

if <BE1> then <S1> else if <BE2> then <S2> else <S3>

are permissible where <BE1>, <BE2> are Boolean expressions and <S1>, <S2>, <S3> are Algol statements which satisfy the restrictions already mentioned. We shall refer to such statements as nested conditional statements.

Example 1

```
if x<y then n:=n+1 else if x=y then goto L
```

Example 2

```
if a=b then print 'equality' else if a=0 then
                          goto out else y:= sqrt(a+b+c)
```

8.2.4 Conditional Arithmetic Expressions

A conditional arithmetic expression has the form

if <Boolean expression>then<arithmetic expression>else<arithmetic expression>

and is a means for selecting one of two arithmetic expressions. Its use will be obvious from the following examples.

Example 1

```
y:= if a<b then 2.5 else a+b+c
```

Example 2

```
Sum:= Sum+(if n=0 then term/2 else term)
```

Example 3

```
print if a=b then a else 0.5×(a+b)
```

N.B. Never follow an arithmetic operator immediately by if. Insert rounded brackets as in Example 2.

8.2.5 Conditional Designational Expressions

A choice between two labels can likewise be made by means of a conditional designational expression which has the form

if <Boolean expression> then <label> else <label>.

Example 1

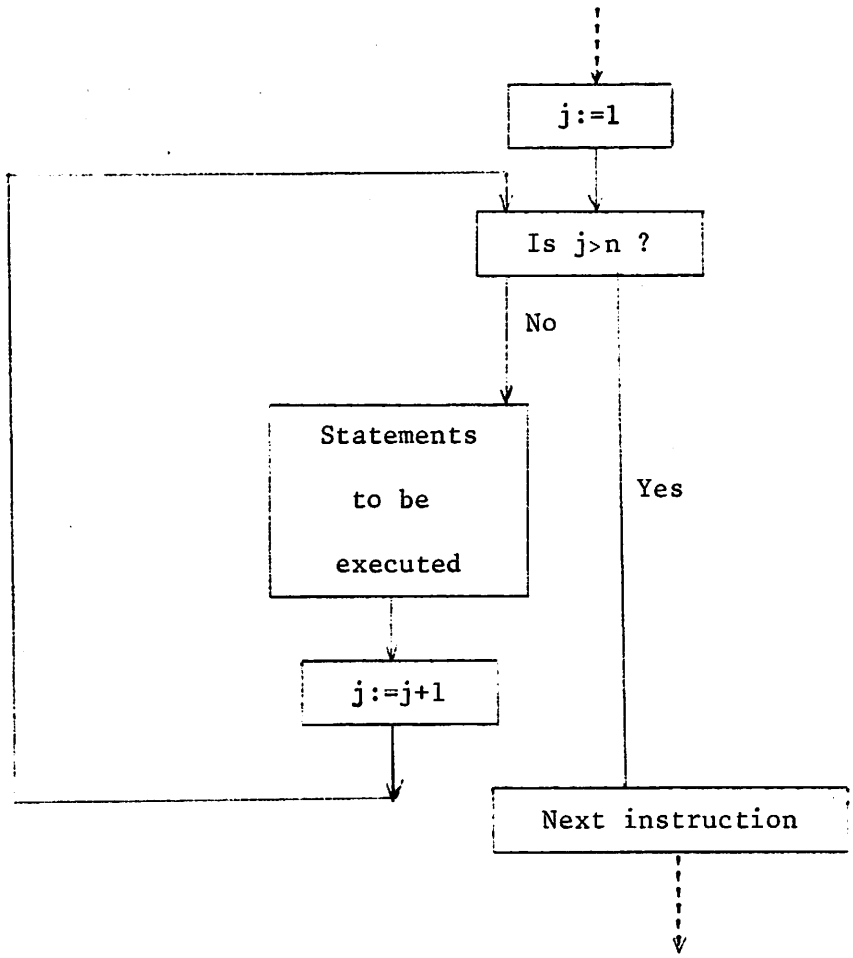
```
goto if abs(x-y)<10-6 then L1 else again
```

Example 2

```
goto if a≥b then out else repeat
```

9.0 For-statements

One of the most useful and powerful statements in Algol is the for-statement. Among other things it enables us to "cycle" i.e. to repeat a number of statements using different values of a variable. The need to do this occurs fairly frequently in programming. Consider, for example, a flow diagram containing the following section



in which a number of statements are executed for j = 1,2,3,--------, n.

Every for-statement has the form

for <identifier>:= <for-list> do <statement>

where

(i) <identifier> is any identifier of type integer or type real declared in the program. The corresponding variable is referred to as the controlled variable

(ii) <statement> is an arbitrary Algol statement called the controlled statement

(iii) <for-list> is a list of <for-list element>s separated by commas and each <for-list element> takes one of the forms

(a) <arithmetic expression>

(b) <arithmetic expression> step <arithmetic expression> until <arithmetic expression>

(c) <arithmetic expression> while <Boolean expression>

We illustrate by example.

Example 1 The Statement

for x:= 1.3,5.7,2.9,11.8,-3.6 do <statement>

means "execute the controlled statement (it may be a compound statement) firstly with x=1.3, then with x=5.7, then with x=2.9,---------- and finally with x=-3.6 ". In this case there are five for-list elements, each of which is an arithmetic expression.

Example 2 The statement

for y:= a+b,(t+q)↑5,cos(x+7.3) do S:=S+y×y

will add, to the value of S, the value of

$(a+b)^2 + (t+q)^{10} + \cos^2(x+7.3)$

Example 3 The statement

for j:=1 step 1 until 50 do <statement>

means "execute the controlled statement firstly with j=1, then with j=2, then with j=3, ---------- and finally with j=50. In this case there is only one for-list element namely

1 step 1 until 50

Example 4 The statement

for kappa:= 3 step 2 until 26 do <statement>

means "execute the controlled statement with kappa taking the values 3,5,7,9,11,----,25". Note that, as soon as the value of the controlled variable (in this case kappa)becomes greater than the value of the "final limit" (in this case 26), execution of the controlled statement ceases and control is transferred to the next statement in the program.

Example 5 The statement

for j:= 30 step - 7 until 0 do <statement>

means "execute the controlled statement with j taking the values 30, 23, 16, 9, 2".

Example 6 The statements

```
S:=0;
for k:=1 step 1 until n do begin
                              read x;
                              S:= S+x
                           end
```

would cause n numbers to be read in and the value of their sum assigned to S.

Example 7 The statement

for x:= a+b step 3×delta until (a+b)↑3 do <statement>

is quite permissible. Once again there is one for-list element of the form

<arithmetic expression> step <arithmetic expression>
until <arithmetic expression>

real step : binary form is rounded up or down slightly so increase the number after until to guarantee proper termination of the loop

Example 8 The statement

```
for x:= 1.0 step 0.01 until 3.005 do <statement>
```

is better than the statement

```
for x:= 1.0 step 0.01 until 3.0 do <statement>
```

(why?).

Example 9 The statements

```
j:=0;
for j:=j+1 while j≤50 do  statement
```

have the same effect as the statement

```
for j:=1 step 1 until 50 do <statement>
```

i.e. the controlled statement is executed with j=1, 2, 3, --------, 50.

Example 10 The statement

```
for k:= 2,3,k+2 while k×k ≤ N do <statement>
```

would cause the controlled statement to be executed

(i) with k=2

(ii) with k=3

(iii) with k=5,7,9,-------- as long as $k^2 \not> N$

Note carefully that, in this case, there are three for-list elements and even if N were equal to 1 the controlled statement would still be executed with k=2 and with k=3.

Example 11 The statement

```
for x:=0 step 1 until 10 step 10 until 100 do <statement>
```

is incorrect. It should be

for x:=0 step 1 until 9, 10 step 10 until 100 do <statement>

i.e. it is a statement containing two for-list elements each of which is of the form

a step b until c

Let us now consider some complete programming examples in which for-statements are used.

Example 12 Tabulate $\frac{1}{6}n(n+1)(2n+1)$ for n=0(1)50.

```
begin
      integer n;
      real y;
      for n:=0 step 1 until 50 do
                  begin
                        y:= n×(n+1)×(2×n+1)/6;
                        print n,SAMELINE,y
                  end
end
```

Example 13 Tabulate $2\sin x + 3.165\, x^2\, e^x$ for x=0(0.1)3.5

```
begin
      real x,y;
      for x:=0 step 0.01 until 3.505 do
                  begin
                        y:=2×sin(x)+3.165×x×x×exp(x);
                        print x,SAMELINE,y
                  end
end
```

Example 14 Given on cards is a value for the integer n followed by n values for $x_1, x_2, x_3, \text{-----}, x_n$. Write a program to evaluate and print $\bar{x}$ and σ where

$$\bar{x} = \frac{1}{n}\Sigma x_i$$

$$\sigma^2 = (\frac{1}{n}\Sigma x_i^2) - \bar{x}^2$$

```
begin
      integer j,n;
      real x, S1, S2, xbar, sigma;
      read n;
      S1:=S2:=0;
      for j:=1 step 1 until n do
                  begin
                        read x;
                        S1:=S1+x;
                        S2:=S2+x×x
                  end;
      xbar:= S1/n;
      sigma:= sqrt(S2/n-xbar↑2);
      print xbar, sigma
end
```

Example 15 Write a program to calculate π, approximately, by summing the first 500 terms of the series

$$\frac{\pi}{4} = 1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \frac{1}{11} \quad - - - - -$$

```
begin
      real S, pi;
      integer j, p;
      S:=0;
      p:=1;
      for j:=1 step 1 until 500 do
            begin
                  S:=S+p/(2×j-1);
                  p:= -p
            end;
      pi:= 4×S;
      print pi
end
```

Example 16 Given on cards are values for n, a_1, b_1, c_1, a_2, b_2, c_2, ------, a_n, b_n, c_n. Write a program to evaluate each of the quadratic functions $a_ix^2 + b_ix + c_i$ for x= 1(1)10.

```
begin
      integer j,x,n;
      real a,b,c,y;
      read n;
      for j:=1 step 1 until n do
      begin
            read a,b,c;
            for x:=1 step 1 until 10 do
            begin
                  y:= a×x×x+b×x+c;
                  print x, SAMELINE, y
            end
      end
end
```

Example 17 A value for P is given on cards. Write a program to calculate and print the number of lattice points inside the circle $x^2 + y^2 = P$. [N.B. A point (h,k) is a lattice point if and only if h,k are integers].

```
begin
      integer k,S,P,T;
      read P;
      if P<0 then begin
                        print 'sorry, P is negative';
                        goto out
                  end;
      T:= entier(sqrt(P+0.1));
      S:= 0;
      for k:=1 step 1 until T do S:= S+2×entier(sqrt(P-k×k+0.1));
      S:= 2×S+4×T+1;
out:end
```

9.1 Value of Controlled Variable on Exit from a For-statement

There are two ways in which an exit can be made from a for-statement, namely

(i) by exhaustion of the for-list i.e. the controlled statement has been executed under the action of every for-list element.

(ii) by means of a goto-statement within the body of the controlled statement (remember the controlled statement may be a compound statement).

The question then arises - what is the value of the controlled variable on exit from the for-statement? The answer is simple - if the exit has occurred because of exhaustion of the for-list then the controlled variable is undefined; if, on the other hand, the exit has occurred as a result of a goto-statement within the controlled statement then the controlled variable has the value it had just before exiting from the for-statement.

10.0 Arrays and Subscripted Variables

Subscripted variables enable us to represent many quantities with one name. For example we are accustomed in mathematics to writing, say

$x_1, x_2, x_3, \text{----------}, x_{100}$

to represent 100 quantities rather than representing them by 100 different variable names, say

x, y, willy, pip, quant, ----------

Similarly in mathematics we are accustomed to double suffix notation in which two subscripts are used. For example we may find it convenient to represent 12 quantities in the form of a two-dimensional array or matrix

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \end{bmatrix}$$

in which the first subscript indicates the row in which the quantity is located and the second subscript the column.

Algol 60 permits a similar notation except that the subscripts are enclosed in square brackets and separated by commas.

Example 1 To refer to the fifth element in the one-dimensional array $x_1, x_2, x_3, \text{--------}, x_{20}$ we write x[5].

Example 2 To refer to the element in the second row and fourth column of the matrix

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} & a_{14} & a_{15} \\ a_{21} & a_{22} & a_{23} & a_{24} & a_{25} \\ a_{31} & a_{32} & a_{33} & a_{34} & a_{35} \\ a_{41} & a_{42} & a_{43} & a_{44} & a_{45} \end{bmatrix}$$

we write a[2,4].

Example 3 The Algol statement

```
d:= sqrt((x[2]-x[1])↑2+(y[2]-y[1])↑2)
```

corresponds to the mathematical formula

$$d = \sqrt{(x_2-x_1)^2 + (y_2-y_1)^2}$$

Example 4 The Algol statement

```
S:= A[1,1] + A[2,2] + A[3,3]
```

corresponds to the mathematical formula

$$S = A_{11} + A_{22} + A_{33}$$

The following points should be noted with regard to subscripted variable notation:-

(i) Before a subscripted variable is used in an Algol program information must be provided to the compiler by means of an array declaration regarding

(a) the name of the array

(b) the upper and lower bounds for each subscript.

Example 5 The declaration

```
real array x[1:20], y[2:13], willy[1:2,1:3]
```

indicates to the compiler that three arrays are going to be referred to later in the program, namely

(a) the one-dimensional array x_1, x_2, x_3, ----, x_{20}

(b) the one-dimensional array y_2, y_3, y_4, ----, y_{13}

(c) the 2×3 matrix

$$\begin{bmatrix} willy_{11} & willy_{12} & willy_{13} \\ willy_{21} & willy_{22} & willy_{23} \end{bmatrix}$$

and that the elements of these arrays are to be held in real form. It is permissible to omit the basic symbol real but not integer or Boolean.

Example 6 The declaration

```
integer array A[1:10,1:20], BOB[1:10,1:20], C[10:15]
```

declares two 10×20 matrices A and BOB, and a one-dimensional array C_{10}, C_{11}, C_{12}, C_{13}, C_{14}, C_{15}. Note that when arrays have identical subscript bounds the bounds need only be mentioned once e.g. in the above case it is permissible to write

```
integer array A,BOB[1:10,1:20], C[10:15]
```

[In Elliott 4100 Algol this is not only permissible but preferable].

(ii) The name of a subscripted variable must be an identifier which is not being used for some other purpose in the same block.

Example 7 A program commencing

```
begin
      real x,y,willy;
      integer i,j,p;
      array A[1:20], x[1:10,5:15];
        ¦
        ¦
        ¦
        ¦
```

is wrong (why?)

(iii) The upper and lower subscript bounds in an array declaration (and also the subscripts themselves in later statements of the program) may be arithmetic expressions. Normally when an arithmetic expression is used as a subscript, or as a subscript bound, it is of type integer. However, if it is of type real, then it will be evaluated and rounded off to the nearest integer.

Example 8 The declaration

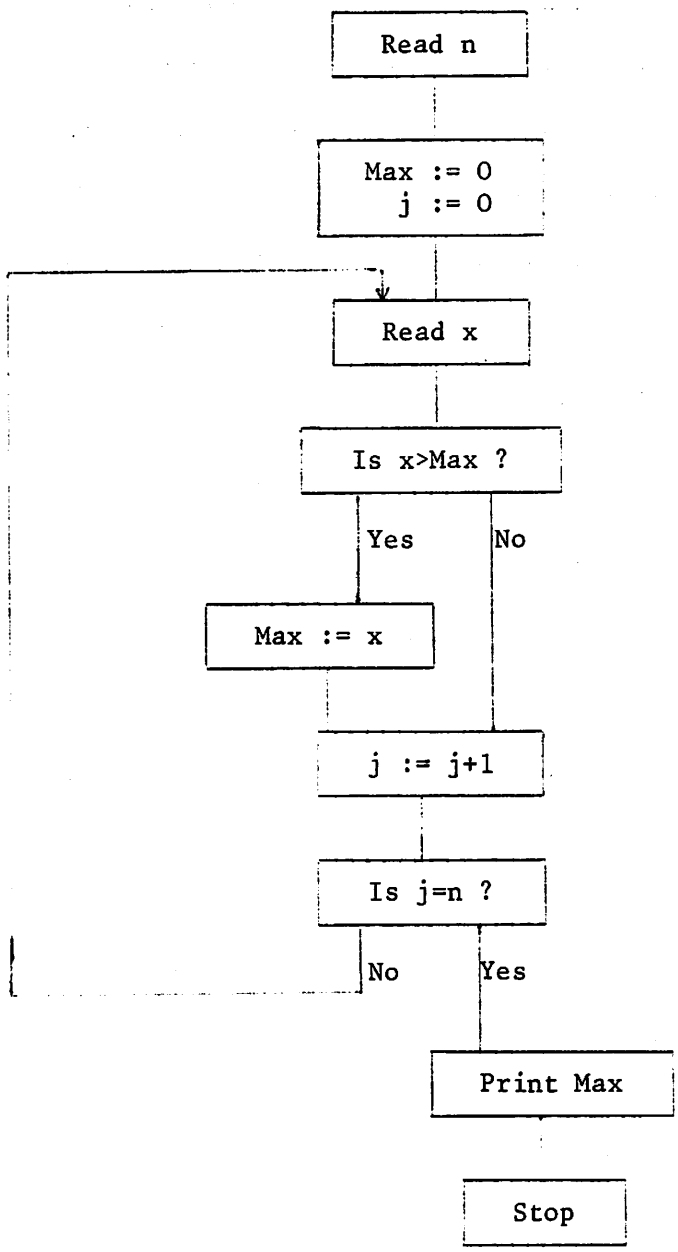
array A[1:n+1], B[m:n], C[1:m-n, 1:m+n]

satisfies the syntactic rules for Algol.

(iv) The elements of an array are held in consecutive storage locations within the store of the computer. This requires a little explanation in the case of matrices (i.e. two-dimensional arrays). In this case the elements are stored row by row i.e. they are held in such a manner that the first subscript varies less rapidly than the second. A similar rule applies to n-dimensional arrays, n>2.

We now consider some complete programs which require the use of arrays. Note that array notation should only be used if necessary. For example the problem:-

Given on cards is an integer n, followed by n positive values x_1, x_2, x_3, --------, x_n. Write a program to find the greatest x-value does not require array notation as can be seen from the following flow diagram:-



Example 9 100 values x_1, x_2, x_3, ---- , x_{100} are given on cards.

Write a program to output the values in reverse order.

```
begin
      integer j;
      array x[1:100];
      for j:=1 step 1 until 100 do read x[j];
      for j:=100 step-1 until 1 do print x[j]
end
```

Example 10 Given on cards is an integer n, followed by n values x_1, x_2, x_3, ------, x_n. Write a program to output the values in reverse order.

It is tempting to begin

```
begin
      integer n,j;
      array x[1:n];
      read n;
        ¦
        ¦
        ¦
        ¦
        ¦
```

but this is incorrect since storage cannot be allocated for the elements of the array x_1, x_2, x_3, ------ , x_n when the value of n, at this stage, has not been read in.

A second attempt might be

```
begin
      integer n,j;
      read n;
      array x[1:n];
        ¦
        ¦
        ¦
        ¦
        ¦
```

but once again we are in trouble over the mixing of declarations and statements - remember that, within a block, all declarations must precede all statements.

The solution is remarkably simple - read in the value for n, then open up a new block before making the array declaration. Thus we obtain

```
begin
      integer n,j;
      read n;
      begin
            array x[1:n];
            for j:=1 step 1 until n do read x[j];
            for j:=n step-1 until 1 do print x[j]
      end
end
```

Example 11 Given on cards is the integer n, followed by the values x_1, y_1, x_2, y_2, ----- , x_n, y_n in that order. Write a program to evaluate and print S where

$$S = x_1y_n + x_2y_{n-1} + x_3y_{n-2} + \text{--------} + x_ny_1$$

```
begin
      integer n; read n;
      begin
            integer j;
            real S;
            array x,y[1:n];
            for j:=1 step 1 until n do
                        read x[j], y[j];
            S:= 0.0;
            for j:=1 step 1 until n do
                        S:=S+x[j]×y[n-j+1];
            print S
      end
end
```

Example 12 Given on cards are values for n, the elements of the n×n matrix A, column by column, and the elements of the n×1 vector x. Write a program to evaluate the vector Ax.

We note that:-

$$Ax = \begin{bmatrix} a_{11} & a_{12} & a_{13} \cdots\cdots a_{1n} \\ a_{21} & a_{22} & a_{23} \cdots\cdots a_{2n} \\ \vdots & & \\ a_{n1} & a_{n2} & a_{n3} \cdots\cdots a_{nn} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix}$$

$$= \begin{bmatrix} a_{11}x_1+a_{12}x_2+\cdots\cdots+a_{1n}x_n \\ a_{21}x_1+a_{22}x_2+\cdots\cdots+a_{2n}x_n \\ \vdots \\ a_{n1}x_1+a_{n2}x_2+\cdots\cdots+a_{nn}x_n \end{bmatrix}$$

so that the elements of Ax are

$$a_{j1}x_1 + a_{j2}x_2 + a_{j3}x_3 + \text{-----} + a_{jn}x_n$$

for j = 1, 2, 3, ------- , n.

This leads to the program:-

```
begin
      integer n; read n;
      begin
            integer i,j,k;
            real S;
            array A[1:n,1:n], x[1:n];
            for j:=1 step 1 until n do
            for i:=1 step 1 until n do read A[i,j];
            for i:=1 step 1 until n do read x[i];
            for j:=1 step 1 until n do
            begin
                  S:= 0.0;
                  for k:=1 step 1 until n do
                                    S:= S+A[j,k]×x[k];
                  print S
            end
      end
end
```

Note how the elements of a matrix can be read in using a "double" for-statement. Of course, if the elements of A had been given on cards, row by row, then a slightly different double for-statement would have been required, namely

```
for i:=1 step 1 until n do
for j:=1 step 1 until n do read A[i,j]
```

11.0 Switches

The arrays which we have encountered so far have either had real, integer or Boolean elements e.g. the declaration

```
integer array x[1:30]
```

declares a one-dimensional array x_1, x_2, x_3, -------- , x_{30} the values of whose elements are to be held in integer form. Occasionally, in Algol, however it is convenient to define a one-dimensional array whose elements are labels! This can be done by means of a switch declaration as indicated in the following examples.

Example 1 The declaration

```
switch S:= ℓ1, next, again, trouble
```

creates a one-dimensional array S such that

S[1] = the label ℓ1

S[2] = the label next

S[3] = the label again

S[4] = the label trouble

Example 2 The declaration

switch pip:= repeat, ℓ5, back, ℓ3, ℓ1

declares a one-dimensional array pip such that

pip [1] = the label repeat

pip [2] = the label ℓ5

pip [3] = the label back

pip [4] = the label ℓ3

pip [5] = the label ℓ1

Note that

(i) A switch declaration has the form

switch <identifier>:= list

where <identifier> (called the name of switch) is any identifier which is not being used for some other purpose in the same block and <list> is a list of label names separated from each other by commas.

(ii) Although a switch is a type of one-dimensional array there is no need to mention upper and lower bounds for the subscript since it is implicit that

{subscript lower bound = 1
subscript upper bound = number of labels listed

(iii) It is advisable to make a switch declaration in the block in a program in which the labels in the switch list occur. Difficulties can arise otherwise. For example

```
begin
      switch SS:= ℓ1,next,ℓ2;
      begin
            integer kappa;
         ℓ1:_______;
            _______;
            _______;
            _______;
         ℓ2:_______;
            _______;
            _______;
            _______;
       next:_______;
            _______;
            _______;
            _______;
            goto SS [kappa]
      end
end
```

is wrong

(iv) A switch list is extremely useful in a program in which it is desired to transfer control to one of several points depending on the value of some expression. Consider, for example, the program

```
begin
      integer k,-------;
      real ------------;
      switch willy:=AA,ℓ1,next, out;
      read k;
      goto willy[k];
   AA:________;
      ________;
      ________;
   ℓ1:________;
      ________;
      ________;
      ________;
 next:________;
      ________;
      ________;
      ________;
  out:________;
      ________;
      ________;
      ________;
      ________;
end
```

Then if the value read in for k is equal to 1, control will be transferred to the instruction labelled willy [1] i.e. to the instruction labelled AA. Similarly if the value read in for k is equal to 2, control will be transferred to the instruction labelled ℓ1 etc.. It is important, of course, that the value read in for k lies between 1 and 4, inclusive. If not then, according to the Revised Report on the Algorithmic Language Algol 60, the effect of the statement

goto willy [k]

is undefined.

(v) As with one-dimensional arrays, when a switch is called into action in a program, the subscript used may be an arithmetic expression. For example, the statement

if x<0 then goto SS[2×k+j-1]

is permissible in a program.

Example 3 A manufacturing firm produces five types of office cabinets. The cost of a cabinet to a purchaser depends on whether he is regarded by the firm as belonging to class 1 customers, class 2 customers or class 3 customers and is given by the following table:-

	Class 1	Class 2	Class 3
type 1	£30	£31	£35
type 2	£42	£44	£49
type 3	£43	£47	£51
type 4	£38	£41	£44
type 5	£40	£42	£45

On a card is given the class number for an individual followed by the number of cabinets of each type which he has purchased. Write a program to calculate his total bill.

```
begin
      integer j,N1,N2,N3,N4,N5,cost;
      switch p:= class1,class2,class3;
      read j,N1,N2,N3,N4,N5;
      goto p[j];
class1:cost:= 30×N1+42×N2+43×N3+38×N4+40×N5;
      goto out;
class2:cost:= 31×N1+44×N2+47×N3+41×N4+42×N5;
      goto out;
class3:cost:= 35×N1+49×N2+51×N3+44×N4+45×N5;
  out:print cost
end
```

Example 4 Given on cards are values for an integer i and a real variable x. Write a program to evaluate $P_i(x)$ where

$$P_0(x) = 1$$

$$p_1(x) = x$$

$$p_2(x) = \tfrac{1}{2}(3x^2-1)$$

$$p_3(x) = \tfrac{1}{2}x(5x^2-3)$$

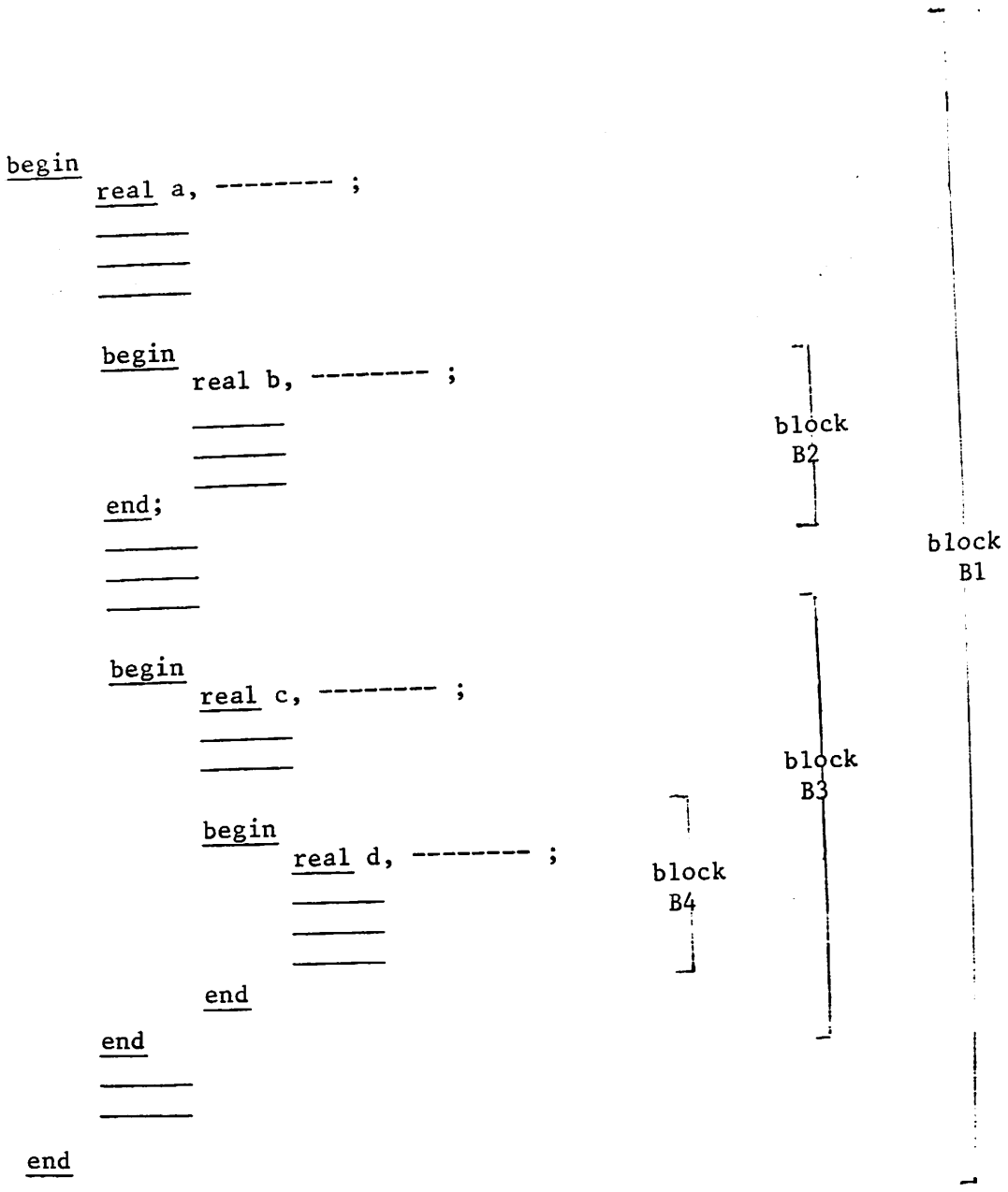
$$p_4(x) = \tfrac{1}{8}(35x^4-30x^2+3)$$

```
begin
      real x,p;
      integer i;
      switch w:= p0,p1,p2,p3,p4;
      read i,x;
      goto w[i+1];
   p0:p:=1; goto output;
   p1:p:=x; goto output;
   p2:p:= (3×x×x-1)/2; goto output;
   p3:p:=x×(5×x×x-3)/2; goto output;
   p4:p:=(35×x↑4-30×x×x+3)/8;
 output:print i, SAMELINE, x,p
end
```

12.0 Algol Block Structure - The Scopes of Quantities

Every quantity occurring in an Algol program has a scope **i.e.** a region of the program in which the quantity exists and can be referred to by its identifier. Outside the scope the quantity is non-existent and cannot be used; its identifier is either meaningless or refers to some other quantity.

Let us consider the scopes of quantities in an Algol program which possesses quite a complicated block structure, say



We first introduce the concept of a sub-block by stating that the block B is a sub-block of the block A if B is contained in A. For example, in the above diagram, block B2 is a sub-block of block Bl, block B4 is a sub-block of block B3 etc..

We are now in a position to define the scopes of the quantities which occur in an Algol program. We shall assume initially that there are no clashes of identifiers in a block and its sub-blocks i.e. the same identifier has not been used more than once as a variable or label name in a block and one of its sub-blocks.

If this is the case, then, if an identifier occurs in a declaration at the start of a block B or as a destination label within block B, the quantity corresponding to the identifier exists throughout B and its sub-blocks but not outside B. The quantity is said to be local to the block B and global to its sub-blocks.

On the other hand, if an identifier which is being used as a variable or label name within a sub-block is identical to a variable or label name in a surrounding block then, while the sub-block is being executed the quantity corresponding to the global variable is suppressed (not lost) and any use of the identifier refers to the local quantity. On exit from the sub-block the value of the global quantity becomes available again and the value of the local quantity is destroyed i.e. it is undefined if the sub-block is re-entered.

The scopes of the various quantities in the previous diagram can now be investigated and it is relatively easy to complete the following table

	BlockB1	BlockB2	BlockB3	BlockB4
a	L	G	G	G
b	X	L	X	X
c	X	X	L	G
d	X	X	X	L

where L represents a local quantity, G a global quantity and X indicates that the quantity does not exist in the block in question.

Example 2 The print-statement in the following program is wrongly placed (why?)

```
begin
      real x,y,u;
      integer i,j,k;
      begin
            real u,willy,w;
            ______
            ______
            ______
            ______
            ______
            ______
      end;
      print w
end
```

Example 3 Consider the following program

```
begin
      real x;
      _______
      _______
      begin
            real x;
            _______
            _______
      end;
      _______
      _______
      _______
end
```

Then, while control is in the sub-block, the value of the global variable x is suppressed (but not lost) and any use of the identifier x in an instruction will refer to the local variable. On exit from the sub-block the value of the local variable x is destroyed and the value of the global variable x is again available to the programmer.

Example 4 The following program is invalid

```
begin
      real x;
      ______
      ______
      begin
            real y;
            ______
            ______
         x:_____
            ______
            ______
            print x;
            ______
            ______
      end
      ______
      ______
end
```

since the global real variable x is suppressed within the sub-block, because of the local label x, and therefore it does not make sense to attempt to print out its value. The difficulty could be avoided by renaming the label x in the sub-block.

Example 5 The program

```
begin
      real x;
      ______
      ______
      begin
            real y;
            ______
            ______
         L:_____
            ______
      end;
      ______
      ______
      goto L;
      ______
      ______
end
```

is wrong since the label L does not exist outside the sub-block and therefore the statement goto L is impermissible. Note that this is an illustration of the often quoted rule - never try to jump into the middle of a block from a surrounding block. It is, of course, permissible to jump out of a block into a surrounding block.

One final point with regard to blocks and block structure - within any block it is forbidden to use the same identifier for more than one purpose. For example it is impermissible to declare an integer pip and an array pip[1:50] in the same block. It is also impermissible to declare a real variable L1 and to use a label L1 in the same block. This important restriction is often expressed by the cryptic phrase "the identifier identifies".

13.0 Comments

The practice of inserting explanatory notes and comments in a program is to be recommended especially if the program is going to be used by others. Such explanatory notes, of course, are for the human reader only and are ignored by the Algol compiler.

There are two ways of inserting comments into an Algol program

(i) Immediately after a begin or a semi-colon write the basic symbol comment, follow it by the desired comments and terminate the information by a semi-colon.

(ii) Use the fact that any remarks written between end and the next semi-colon, end or else are ignored by the compiler.

Example 1 The following program contains several comments:-

```
begin
      comment This is a program which reads in n
              values x[1] to x[n] and outputs them
              in reverse order;
      integer n;
      comment Read in the value of n;
      read n;
      begin
              array x[1:n]; integer j;
              comment Read in values now;
              for j:=1 step 1 until n do read x[j];
              comment Print out values in reverse order;
              for j:=n step -1 until 1 do print x[j]
      end inner-block
end specimen program;
```

14.0 Boolean Variable and Expressions

In addition to real and integer quantities Algol permits the use of Boolean (or logical) quantities. However whereas real and integer quantities take on numerical values in a program a Boolean quantity can only assume a logical value i.e. the value of a Boolean quantity is either "true" or "false".

Example 1 The assignment statements

```
flag:= true;
   q:= false
```

set the Boolean variables flag and q equal to "true" and "false" respectively. Note that true and false are basic symbols in Algol.

Example 2 The assignment statement

```
pip:= a+b<c
```

assigns to the Boolean variable pip the value "true", if a+b<c, and the value "false" if a+b≥c. Here, of course, a,b and c are quantities of type real or integer

Example 3 The assignment statement

```
mark:= i+j = 3×(j+k+7)
```

assigns to the Boolean variable mark the value "true", if i+j = 3(j+k+7), and the value "false" if i+j ≠ 3(j+k+7). In this case i,j,k are probably quantities of type integer (why?)

Example 4 The conditional statement

```
if test then y:= x+1
```

does not make sense if test is of type real or integer. However if test is a Boolean variable which has already been assigned a value ("true" or "false") then the statement is syntactically correct.

Example 5 The statement

flag:= if x>y then i=j+1 else p>q

where flag is a Boolean variable and x,y,i,j,p and q are of type real or integer is equivalent to the statement

if x>y then flag:= i=j+1 else flag:= p>q

Note that the expression

if x>y then i=j+1 else p>q

is called a conditional Boolean expression (c.f. Sections 8.2.4 and 8.2.5)

Example 6 The statement

if if x then y else z then goto next

is syntactically correct if x,y,z are Boolean variables.

14.1 Logical Operators

We have already seen how complicated arithmetic expressions can be formed by using the arithmetic operators +,−,×,/,÷,↑ and the rounded brackets (and) . In a similar manner complicated Boolean expressions can be formed by using the logical operators ¬ ,∧,∨,≡ and the rounded brackets (and) . Let us consider each logical operator in turn.

(a) The Negation Operator ¬ ("not")

If b is any Boolean expression then ¬ b is "true" when b is "false" and ¬ b is "false" when b is "true".

b	¬ b
true false	false true

(b) The Logical Operator ∧ ("and")

If b and c are Boolean expressions then b∧c is "true" if and only if b,c are both "true".

b	c	b∧c
false false true true	false true false true	false false false true

(c) The Logical Operator V ("or")

If b and c are Boolean expressions then bVc is "false" if and only if b,c are both "false".

b	c	bVc
false false true true	false true false true	false true true true

(d) The Logical Operator ⊃ ("implies")

If b and c are Boolean expressions then b⊃c is "false" if and only if b is "true" and c is "false".

b	c	b⊃c
false	false	true
false	true	true
true	false	false
true	true	true

(e) The Logical Operator ≡ ("is equivalent to")

If b and c are Boolean expressions then b≡c is "true" if and only if b and c are both "true" or both "false".

b	c	b≡c
false	false	true
false	true	false
true	false	false
true	true	true

As with arithmetic expressions rounded brackets, (and), may be freely inserted where necessary or where it improves readability to do so. There is also, of course, an order of precedence in evaluating a Boolean expression, namely

(i) anything in rounded brackets

(ii) ¬

(iii) ∧

(iv) ∨

(v) ⊃

(vi) ≡

and, in the cases of operators of equal precedence, the order of precedence is from left to right.

Example 1 The following are permissible involving Boolean expressions:-

(i) Test:= a>b V i≠j+17

(ii) flag:= cos(x)<0.2 Λ (x+y)↑3 ≥ 1.6

(iii) marker:=¬ (a+b>c)

(iv) if m>60 Λ (p>50 V c>50 V e>50) then n:=n+1

(v) signal:=¬ flag Λ Test V marker

(vi) indicator:= x+y>6 V p+q<17 ≡ L>19 Λ F<a+b

Note that the insertion of superfluous rounded brackets would make (v) and (vi) much easier to read, namely

(v) signal:= ((¬ flag)ΛTest)V marker

(vi) indicator:= ((x+y>6)V(p+q<17))≡((L>19Λ(F<a+b))

Example 2 On cards is a list of candidate numbers, each number being followed by the candidate's marks in mathematics, physics, chemistry and German. The data values are terminated by -1. Write a program to print out the number of each candidate who scores more than 40 in German and more than 50 in at least two other subjects.

```
     begin
          integer cn,m,p,c,g;
       L1:read cn;
          if cn<0 then goto out;
          read m,p,c,g;
          if g>40Λ((m>50Λp>50)V(m>50Λc>50)
                        V(p>50Λc>50))then print cn;
          goto L1;
out:end
```

Example 3 On cards is a list of candidate numbers, each number being followed by the candidate's marks in mathematics, physics, chemistry, German and English. The data values are terminated by -1. A candidate belongs to

(i) category A if he scores more than 50 in mathematics, more than 60 in either physics or chemistry, and at least 40 in English.

(ii) category B if he scores more than 75 in English, more than 60 in German, and at least 50 in one other subject.

(iii) category C if he scores more than 75 in mathematics, at least 60 in both physics and chemistry, at least 50 in English and more than 40 in German.

Write a program to find

(a) the number of candidates in each category.

(b) the number of candidates who belong to both category A and category B.

(c) the number of candidates in either category A or category C (or both).

(d) the number of candidates who belong to all three categories.

```
begin
      integer m,p,c,g,e,n1,n2,n3,n4,n5,n6,cn;
      Boolean catA,catB,catC;
      n1:=n2:=n3:=n4:=n5:=n6:=0;
again:read cn;
      if cn<0 then goto out;
      read m,p,c,g,e;
      catA:= (m>50)∧(p>60∨c>60)∧(e≥40);
      catB:= (e>75)∧(g>60)∧(m≥50∨p≥50∨c≥50);
      catC:= (m>75)∧(p≥60)∧(c≥60)∧(e≥50)∧(g>40);
      if catA then n1:=n1+1;
      if catB then n2:=n2+1;
      if catC then n3:=n3+1;
      if catA ∧ catB then N4:=n4+1;
      if catA ∨ catC then n5:=n5+1;
      if catA ∧ catB ∧ catC then n6:=n6+1;
      goto again;
  out:print n1,SAMELINE,n2,n3,n4,n5,n6
end
```

15.0 Own Variables

We already know, from our discussions in Section 12.0 on block structure, that, whenever control is transferred out of a sub-block, the values of variables local to the sub-block are destroyed and are therefore undefined on re-entry to the sub-block. This could lead to difficulties in certain situations (it is not easy to think of such a situation) and a means is therefore provided in Algol by which the values of variables local to a sub-block are preserved on exit from the sub-block and become available again on re-entry to the sub-block.

How this is done is quite simple - before the basic symbol real, integer or Boolean in a declaration in a sub-block write the basic symbol own.

Example 1 Consider the program

```
begin
      real alpha, pip,willy;
      integer omega;
      ____________
      ____________
      ____________
      ____________
      ____________
      ____________
      begin
            real a,b,x;
            own real u,v;
            integer p,q;
            own integer y,z,i,j;
            array KAPPA[1:10,1:20];
            own real array C[1:50];
            own integer array D[1:17];
            ____________________
            ____________________
            ____________________
            ____________________
      end;
      _________
      _________
      _________
      _________
end
```

Then, on exit from the sub-block, the values of u,v,y,z,i,j,C_1,C_2, ------, C_{50},D_1,D_2, ------, D_{17} would be preserved and would become available again on re-entry to the sub-block. The values of the other variables local to the sub-block would be destroyed on exit from it.

16.0 Procedures

Quite often it is necessary to repeat the same series of Algol statements at a number of different places in a program. It is possible, of course, to write out the necessary statements each time they are required but doing so wastes both programming time and machine storage space. It would therefore be extremely convenient if it were possible to write out the relevant statements only once in the program and to be able to give them a "name" by which they could be brought into action wherever required. In Algol this is made possible by the use of procedures.

We consider first "ordinary" procedures and later "function" or "type" procedures.

16.1 Ordinary Procedures

Consider the following program:-

```
begin
      real a,b,x,y.----------;
      integer i,j,-----------;
      array ----------------;
      integer array ---------;
      switch SS:= -----------;
      _______________
      _______________
      _______________
      if a<b then y:=x+1 else y:=x↑5+x
      _______________
      _______________
      if a<b then y:=x+1 else y:=x↑5+x
      _______________
      _______________
      _______________
      if a<b then y:=x+1 else y:=x↑5+x
      _______________
      _______________
      _______________
      _______________
      if a<b then y:=x+1 else y:=x↑5+x
      _______________
      _______________
      _______________
      _______________
end
```

It is obviously a tedious business to write out repeatedly the statement

```
if a<b then y:=x+1 else y:=x↑5+x
```

and this can be avoided by giving the statement a "name" in a procedure declaration and then writing this name wherever it is required to perform the statement in question. The program now looks like:-

```
begin
      real a,b,x,y,--------;
      integer i,j,---------;
      array ---------------;
      integer array -------;
      switch SS:= ---------;
      procedure val;
        if a<b then y:=x+1 else y:=x↑5+x;
      ________________
      ________________
      ________________
      val;
      ________________
      ________________
      val;
      ________________
      ________________
      ________________
      val;
      ________________
      ________________
      ________________
      val;
      ________________
      ________________
      ________________
      ________________
end
```

Note that

(i) procedure declarations in a block follow the non-procedure declarations.

(ii) the name of the procedure in the above case is val. Procedure names must conform to the usual rules for identifiers.

(iii) the body of the procedure declaration in the above case is the single statement

```
if a<b then y:=x+1 else y:=x↑5+x
```

If the body of a procedure consists of several statements then the statements must be enclosed between the statement brackets begin and end.

(iv) the procedure is brought into action later in the program (i.e. the body of the procedure is executed) merely by writing the procedure statement

val

which is the name of the procedure.

Example 2 Write a procedure SWAP to interchange the values of the variables a,b when it is called into action.

```
begin
      real a,b,----------;
      integer ----------;
      array ----------;
      procedure  SWAP;
      begin
            real T;
            T:=a; a:=b; b:=T
      end;
      _____________
      _____________
      _____________
      _____________
      SWAP;
      _____________
      _____________
      _____________
      SWAP;
      _____________
      _____________
      _____________
      SWAP;
      _____________
      _____________
      _____________
end
```

Note that, in this case, we have declared a real variable T which is local to the body of the procedure.

Example 3

```
begin
      real a,b,c,v,----------;
      integer ---------------;
      procedure MAX;
      begin
            v:=a;
            if b>v then v:=b;
            if c>v then v:=c
      end;
      ____________
      ____________
      ____________
      MAX;
      ____________
      ____________
      ____________
      ____________
      MAX;
      ____________
      ____________
      ____________
      ____________
      ____________
      MAX;
      ____________
      ____________
end
```

Example 4

```
begin
      integer n; read n;
      begin
            real sum, ----------;
            integer ------------;
            array x[1:n], ----------;
            procedure vecsum;
            begin
                  integer j;
                  sum:=0.0;
                  for j:=1 step 1 until n do
                                    sum:= sum+x[j
            end vecsum;
            ________
            ________
            ________
            vecsum;
            ________
            ________
            vecsum;
            ________
            ________
            ________
            vecsum;
            ________
            ________
      end
end
```

16.2 Procedures with Formal Parameters

As they stand the procedures which appeared in the last section are too restrictive to be of great value to a programmer. For example the procedure SWAP will only interchange the values of the real variables a and b, and the procedure vecsum will only sum the values of the elements of the array x_1, x_2, x_3,--------, x_n. It would be much more convenient if SWAP could be made to interchange the values of any two real variables, declared in the program, and if vecsum could be made to sum the values of the elements of any one-dimensional array declared in the program. Formal parameters enable us to achieve this kind of flexibility.

Example 1

```
begin
      real a,b,x,p,q,willy,T,---------;
      integer i,kappa,----------;
      array alpha[1:50],----------;
      procedure SWAP(x,y);
      real x,y;
      begin
            real T;
            T:=x; x:=y; y:=T
      end SWAP;
      _________
      _________
      _________
      SWAP(a,b);
      _________
      _________
      _________
      SWAP(q,willy);
      _________
      _________
      _________
      SWAP(x,q);
      _________
      _________
      _________
      _________
end
```

Note that

(i) Two formal parameters, x and y, are used in the procedure SWAP, since our aim is to be able to interchange the values of any two real variables, in the program, by calling SWAP into action.

(ii) A formal parameter has no independent existence - unlike a declared quantity. It is merely a representation of an actual parameter which will replace it when the procedure is called into action. Thus, for example, when we write

SWAP(a,b)

in the above program, the body of the procedure SWAP is obeyed with the formal parameter x replaced by the real variable name a, and the formal parameter y replaced by the real variable name b. Essentially then the following statements are executed

```
begin
      real T;
      T:=a; a:=b; b:=T
end
```

(iii) Since formal parameters are replaced by actual parameters, whenever a procedure is called into action, it is necessary to specify, in the procedure declaration, the type of quantity represented by each formal parameter. In the above example since the formal parameters x,y represent real variables we merely write

```
real x,y;
```

in the indicated position. Note that this is not a declaration of actual variables. It is a type specification of formal parameters and indicates that x,y represent real variable names and will be replaced by such, whenever the procedure is called into action. In later examples we shall see the following specifiers used:-

string	label
real	switch
integer	procedure
Boolean	real procedure
array	integer procedure
integer array	Boolean procedure
Boolean array	

By way of illustration a procedure commencing:-

```
procedure SEARCH(A,m,n,L);
integer m,n;
array A;
label L;
begin
   ¦
   ¦
   ¦
   ¦
   ¦
   ¦
```

is such that, when it is called into action in a program, the formal parameters m,n will be replaced by integer expressions, the formal parameter A will be replaced by the name of a real array and the formal parameter L will be replaced by the name of a label.

(iv) No confusion is caused in the example using SWAP by having a formal parameter x and a real variable x. For example, when the procedure call

```
SWAP(x,q)
```

is executed, the formal parameter x, in the procedure declaration, is replaced by the real variable name x declared in the program.

(v) The following calls of the procedure SWAP are impermissible (why?):-

```
SWAP (a,kappa)
SWAP (2.75,x)
SWAP (a, x+q)
SWAP (i,kappa)
```

(vi) No confusion is caused by writing a procedure call, such as

```
SWAP(T,a)
```

since, when the body of the procedure is executed with the formal parameters x,y replaced by T,a, the local variable T is renamed so that it does not clash with the actual parameter T, declared in the program i.e. the following statements are executed:-

```
begin
      real T1;
      T1:=T; T:=a; a:=T
end
```

Here we have renamed the local variable T, in the body of the procedure, so that it becomes T1.

Example 2

```
begin
      real p,q,r,u,v,w,x,y,z,----------;
      integer ----------;
      procedure MAX(a,b,c,u);
      real a,b,c,u;
      begin
            u:=a;
            if b>u then u:=b;
            if c>u then u:=c;
      end MAX;
      ________
      ________
      ________
      MAX(p,q,r,x);
      ________
      ________
      MAX(1.6,3.2,4.9,z);
      ________
      ________
      MAX(p+2×q,u×u+v×v,19.5,r);
      ________
      ________
      ________
      MAX(u,exp(p+0.6),(y+z)↑17,w);
      ________
      ________
      ________
end
```

Example 3

```
begin
      integer n; read n;
      begin
            real S,Sum, Total,Val,----------;
            integer i,j,kappa,num,----------;
            array x,y[1:n], pip[1:50], q[1:2×n],
                        A[1:n,1:n], B[1:20,1:30];
            procedure vecsum (A,n,y);
            array A;
            integer n;
            real y;
            begin
                  integer j;
                  y:=0.0;
                  for j:=1 step 1 until n do y:=y+A[j]
            end vecsum;
              ¦
              ¦
              ¦
              ¦
              ¦
              ¦
```

Later in the program we might write

```
vecsum(pip,50,Sum)
```

and this would cause the following instructions to be executed:-

```
begin
      integer j;
      Sum:=0.0;
      for j:=1 step 1 until 50 do Sum:=Sum+pip[j]
end
```

so that Sum would be given the value of $pip_1 + pip_2 + pip_3 + \text{------} + pip_{50}$.

Similarly we might write

```
vecsum (q, 2×n, Total)

vecsum (x, 20, Val)

vecsum (y, n, S)        etc..
```

Note however that procedure calls of the following types would be impermissible (why?).

```
vecsum (A, n, S)

vecsum (B, 20, Total)
```

Example 4 The following procedure SEARCH searches through the elements A_m, A_{m+1},------, A_n and, if it finds a zero element, a jump to a specified label L takes place.

```
begin
      integer i,kappa,count,----------;
      real --------;
      array x,y[1:50], p[1:80], z[10:20],
                              A,WILLY[1:5,1:25];
      procedure SEARCH (A,m,n,l);
      array A;
      integer m,n;
      label L;
      begin
            integer j;
            for j:=m step 1 until n do
                  if A[j]=0 then goto L
      end SEARCH;
      ----------
      ----------
      SEARCH (x,1,50,out);
      ----------
  out:----------
      SEARCH (z,10,15,next);
      ----------
      ----------
      ----------
      SEARCH (p,i,kappa,trouble);
      ----------
      ----------
trouble:----------
      ----------
      ----------
 next:----------
end
```

Example 5 Consider the following procedure

```
procedure biquad (a,b,c,d,e,t,y);
real a,b,c,d,e,t,y;
y:= (((a×t+b)×t+c)×t+d)×t+e;
```

Then, if later in a program, containing biquad, the procedure call

```
biquad (1.6,3.5,2.7,3.8, -1.5,p,val)
```

was made, the statement

```
val:= (((1.6×p+3.5)×p+2.7)×p+3.8)×p-1.5
```

would be executed.

Similarly, if the procedure call

```
biquad (3.91,-1.6,-8.24,1.65,8.7,cos(exp(x+3)),v)
```

was made, the statement

```
v:= (((3.91×cos(exp(x+3))-1.6×cos(exp(x+3))-8.24)×cos(exp(x+3))+1.65)×
                                                    cos(exp(x+3))+8.7
```

would be executed. Note that, in the second case, the formal parameter t is replaced by the actual expression cos(exp(x+3)) (not its value!) and hence, in evaluating the statement, the expression cos(exp(x+3)) is evaluated four times. For this (and other reasons) we introduce the idea of a value list.

16.3 The Value List - Execution of a Procedure Statement

Immediately before the type specifications of the formal parameters of a procedure it is permissible to create a value list. For example, in setting up the procedure biquad we might have written:-

```
procedure biquad (a,b,c,d,e,t,y);
value a,b,c,d,e,t;
real a,b,c,d,e,t,y;
y:= (((a×t+b)×t+c)×t+d)×t+e;
```

Then, on the procedure being called into action later in the program, effects equivalent to the following take place:-

(i) a block (the so called equivalence block) is created

(ii) for every parameter called by value (i.e. whose name appears on the value list) a corresponding declaration of a local variable is made at the beginning of the equivalence block.

In addition, statements are made which assign to each declared local variable the current value of the actual parameter by which the corresponding formal parameter has been replaced.

(iii) for every parameter called by name (i.e. not by value) a substitution occurs in the procedure body by which the formal parameter is replaced by the actual expression (not its value) which is given in the procedure call. The amended procedure body is then inserted into the equivalence block.

(iv) the equivalence block is executed.

One further point - when an identifier, local to the body of the procedure, clashes with an identifier which is to be inserted as a result of (iii) above, the name of the local quantity is altered before any insertions take place.

Example 1

```
procedure biquad (a,b,c,d,e,t,y);
value a,b,c,d,e,t;
real a,b,c,d,e,t,y;
y:= (((a×t+b)×t+c)×t+d)×t+e;
```

If, later in a program containing biquad, the procedure statement

```
biquad (3.91,-1.6,-8.24,1.65,8.7,cos(exp(x+3)),v)
```

appeared, then its effect would be equivalent to the execution of the following block:-

```
begin
      real a,b,c,d,e,t;
      a:= 3.91;
      b:= -1.6;
      c:= -8.24;
      d:= 1.65;
      e:= 8.7;
      t:= cos(exp(x+3));
      v:= (((a×t+b)×t+c)×t+d)×t+e
end
```

Note that it would be wrong to put y in the value list (why?). This is a specific instance of the general rule - never put the output parameters of a procedure in a value list.

Example 2

```
procedure swap(x,y);
real x,y;
begin
      real T;
      T:=x; x:=y; y:=T
end;
```

Here again it would be wrong to insert the formal parameters x,y in a value list.

Example 3

```
procedure vecsum (A,n,y);
value n;
array A;
real y;
begin
      integer j;
      y:=0;
      for j:= 1 step 1 until n do y:=y+A[j]
end vecsum;
```

If, later in the program, the procedure call

```
vecsum (pip,50,Sum)
```

appeared, then the effect of the procedure call is equivalent to the execution of the following block:-

```
begin
      integer n; n:=50;
      begin
            integer j;
            Sum:=0;
            for j:=1 step 1 until n do
                              Sum:= Sum+pip[j]
      end
end
```

If, however, the array name A had appeared on the value list, thus:-

```
procedure  vecsum (A,n,y);
value n,A;
array A;
   ¦
   ¦
   ¦
   ¦
   ¦
   ¦
```

then the same procedure call would have had an effect equivalent to the execution of the following block (assuming that the one-dimensional array pip has 50 elements):-

```
begin
      integer n;
      array A[1:50];
      n:= 50;
      begin
            integer k;
            for k:=1 step 1 until 50 do A[k]:= pip[k]
      end;
      begin
            integer j;
            Sum:=0;
            for j:=1 step 1 until n do
                                    Sum:=Sum+A[j]
      end
end
```

Note therefore that, if an array is called by value, a copy is made of the actual array. This, of course, takes up valuable storage space and hence arrays should not be called by value unless it is absolutely essential to do this.

Example 4

```
procedure SEARCH (A,m,n,L);
value m,n;
array A;
integer m,n;
label L;
begin
      integer j;
      for j:=m step 1 until n do
                  if A[j]=0 then goto L
end SEARCH;
```

Example 5 Write a procedure SPROD(x,y,n,S,xmax,ymax) to compute, for the two n×1 vectors x,y:-

(i) $S = x_1y_1 + x_2y_2 + \text{----------} + x_ny_n$

(ii) xmax = largest element of vector x

(iii) ymax = largest element of vector y.

```
procedure SPROD(x,y,n,S,xmax,ymax);
value n;
array x,y;
real xmax, ymax, S;
begin
      integer j;
      xmax:= ymax:= -10^50;
      S:=0.0;
      for j:=1 step 1 until n do
      begin
            S:=S+x[j]×y[j];
            if x[j]>xmax then xmax:= x[j];
            if y[j]>ymax then ymax:= y[j]
      end
end SPROD;
```

Example 6 Write a procedure TAB(a,b,h,x,f) which will tabulate the function f(x) for x=a(h)b i.e. for x=a, a+h, a+2h, ------, b-h, b.

```
procedure TAB(a,b,h,x,f);
value a,b,h;
real a,b,h,x,f;
begin
      real t;
      t:= b+0.5×h;
      for x:=a step h until t do
                  print x, SAMELINE, f
end TAB;
```

At first sight this does not seem to achieve the correct result. Consider, however, the procedure call

```
TAB(0.0,3.0,0.001,u,exp(u))
```

which has an effect equivalent to the creation and execution of the following block:-

```
begin
      real a,b,h;
      a:=0.0; b:=3.0; h:=0.001;
      begin
            real t;
            t:= b+0.5×h;
            for u:= a step h until t do
                  print u, SAMELINE, exp(u)
      end TAB
end
```

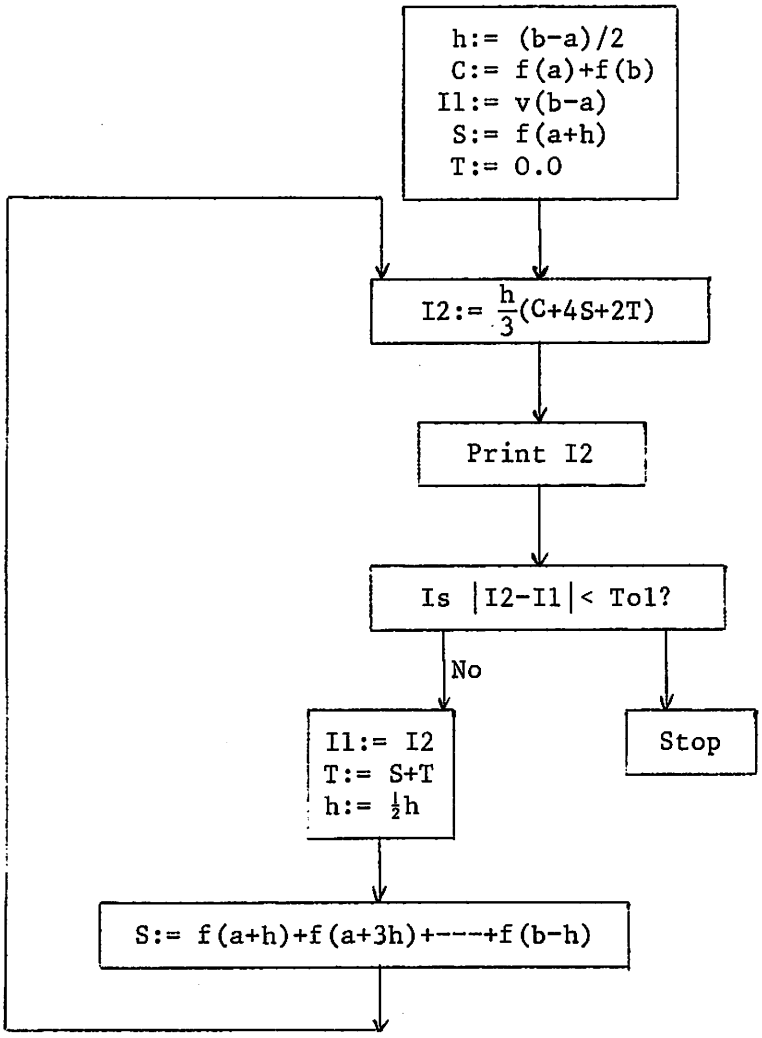
so that e^u would be tabulated for u=0.0(0.001)3.0!! Note that the strategy (called Jensen's Device) is to make the actual parameter replacing f a function of the actual parameter replacing x, and it works because x and f are called by name and not by value.

Example 7[†] The following flow-diagram and procedure describe a method for evaluating

$$\int_a^b f(x)dx$$

by repeated application of Simpson's rule. v is any quantity such that v > max f(x),a≤x≤b.

[†]A knowledge of numerical integration by Simpson's rule is assumed in this example.


h:= (b-a)/2
C:= f(a)+f(b)
I1:= v(b-a)
S:= f(a+h)
T:= 0.0
I2:= h/3(C+4S+2T)
Print I2
Is |I2-I1| < Tol?
No
Stop
I1:= I2
T:= S+T
h:= ½h
S:= f(a+h)+f(a+3h)+---+f(b-h)

```
        procedure SIMPSON(a,b,x,f,v,Tol);
        value a,b,v,Tol;
        real a,b,x,f,v,Tol;
        begin
              real h,C,S,T,I1,I2;
              integer j;
              h:= 0.5×(b-a);
              x:=a;
              C:=f;
              x:=b;
              C:= C+f;
              I1:= v×(b-a);
              x:= a+h;
              S:= f;
              T:= 0.0;
      repeat:I2:= h×(C+4.0×S+2.0×T)/3.0;
              print I2;
              if abs(I2-I1)<Tol then goto out;
              I1:=I2;
              T:= S+T;
              h:= 0.5×h;
              S:= 0.0;
              for j:=1,j+2 while a+j×h<b do
              begin
                    x:= a+j×h;
                    S:= S+f
              end;
              goto repeat;
 out:end SIMPSON;
```

This example will repay careful consideration. Note how the instructions

```
x:=a;
C:=f;
x:=b;
C:=C+f;
```

take effect when the procedure is called into action, say by the instruction

```
SIMPSON(0.0,1.0,t,1.0/sqrt(1.0+t↑3),2.0,0.5₁₀⁻6).
```

16.4 Function Procedures (or how to create your own standard functions)

A function procedure differs from an ordinary procedure in a number of important ways:-

(i) When a function procedure is called into action it produces essentially a single value which is either of type real, integer or Boolean. Depending on the type of value produced by the function procedure, the basic symbol procedure is preceded by either real, integer or Boolean when making the procedure declaration.

(ii) In the body of the procedure at least one assignment statement must occur which has the name of the procedure on the left hand side and which assigns to this (fictitious) variable the single value produced by the procedure. Normally the name of the procedure will not occur on the right hand side of any statement within the procedure body. If it does then we have a recursive procedure (See Section 16.5).

(iii) Unlike an ordinary procedure, a function procedure is called into action by using it, within an expression, in the same way that we use one of the standard functions.

Example 1

```
begin
      real a,b,c,pip,alpha;
      integer i,j,k,----------;
      real procedure radius (x,y);
      value x,y;
      real x,y;
      radius:= sqrt(x×x+y×y);
          ¦
          ¦
          ¦
          ¦
          ¦
          ¦
          ¦
```

Later in the program we might write

```
pip:=radius(b,c)
```

which would give pip the value of $\sqrt{b^2+c^2}$, or

```
alpha:= (3.6+radius(2.5,a))↑6+cos(radius(a,c))
```

which would give alpha the value of $(3.6+\sqrt{(2.5)^2+a^2})^6 + \cos\sqrt{a^2+c^2}$, or

```
c:= radius (radius(a,b),radius(pip,alpha))
```

which would give c the value of $\sqrt{a^2+b^2+pip^2+alpha^2}$

etc..

Example 2

```
begin
      real a,b,c,pip,----------;
      integer n,j,kappa,count, ----------;
      array x,y[1:35], willy[10:60], B,CAT[1:30,1:90];
      real procedure vecsum (A,n);
      value n;
      array A;
      integer n;
      begin
            real S; integer j;
            S:= 0.0;
            for j:=1 step 1 until n do S:=S+A[j];
            vecsum:=S
      end vecsum;
```

Later in the program we might write

```
pip:= vecsum(x,20) + vecsum(y,35)

c:= (vecsum(x,kappa)+a)↑3.652
```

Example 3

```
begin
      real a,b,t,q,willy,s,val,----------;
      integer i,j,kappa,----------;
      real procedure sinh(x);
      value x; real x;
      begin
            real t; t:=exp(x);
            sinh:= 0.5×(t-1.0/t)
      end sinh;
      real procedure cosh(x);
      value x; real x;
      begin
            real t; t:= exp(x);
            cosh:= 0.5×(t+1.0/t)
      end cosh;
        ¦
        ¦
        ¦
        ¦
        ¦
        ¦
```

later in the program we might write:-

```
      willy:= sinh(3×a - 2×b + cos(q))

      val:= sinh(cos(s)) + cosh(sin(s))

        a:= sinh(t×t+q×q)↑6 + cosh(t×t-q×q)↑3.9
```

etc..

Example 4 Write a function procedure to compute

$$y = \begin{cases} 1 + \sqrt{1+x^2} & \text{if } x<0 \\ 1 - \sqrt{1-x^2} & \text{if } x\geq 0 \end{cases}$$

```
real procedure y(x);
value x;
real x;
begin
      real t;
      t:= sqrt(1+x×x);
      if x<0 then y:=1+t else y:=1-t
end y;
```

Example 5 Write a function procedure FAC(n) to compute n! , for n≥1.

```
integer procedure FAC(n);
value n;
integer n;
begin
      integer j,t;
      t:=1;
      for j:=1 step 1 until n do t:=t×j;
      FAC:=t
end FAC;
```

Example 6 Write a function procedure NZERO(A,m,n) to compute the number of zero elements in the m×n matrix A.

```
integer procedure NZERO(A,m,n);
value m,n;
array A;
integer m,n;
begin
      integer i,j,S;
      S:=0;
      for i:=1 step 1 until m do
      for j:=1 step 1 until n do
            if A[i,j]=0 then S:=S+1;
      NZERO:=S
end NZERO;
```

Example 7 Consider the following function procedure:-

```
real procedure Sigma(m,n,j,f);
value m,n;
integer m,n,j;
real f;
begin
      real S;
      S:=0.0;
      for j:=m step 1 until n do S:=S+f;
      Sigma:=S
end Sigma;
```

This, at first glance, seems a fairly trivial calculation. However suppose that the procedure was called into action, in a program, by writing

```
val:= Sigma (1,30,k,k↑2)
```

Then the effect of this instruction would be equivalent to the execution of the following block, followed by the assignment of the value of Sigma to val:-

```
begin
      integer m,n;
      m:=1; n:=30;
      begin
            real S;
            S:=0.0;
            for k:=m step 1 until n do S:=S+k↑2
            Sigma:=S
      end Sigma
end
```

In other words val is assigned the value of $1^2+2^2+3^2+----------+30^2$. Once again, then, we are bringing Jensen's Device into operation by making the actual parameter replacing f a function of the actual parameter replacing j and it works because f,j are not on the value list. Other possible assignment statements which might occur later in the program are:-

```
w:= Sigma (1,n,i,A[i])
```

which would set w equal to the value of $A_1+A_2+----------+A_n$ and

```
spur:= Sigma (1,n,m,C[m,m])
```

which would set spur equal to the value of $C_{11}+C_{22}+----------+C_{nn}$ and

```
RESULT:= Sigma (1,m,i, Sigma(1,n,j, B[i,j]))
```

which would set RESULT equal to the sum of the elements of the m×n matrix B.

16.5 Recursive Procedures

Consider the following procedure† for computing n! for n≥1:-

† c.f. Example 5, Section 16.4

Example 1

```
integer procedure FAC(n);
value n;
integer n;
FAC:= if n=1 then 1 else n×FAC(n-1);
```

This is an example of a recursive procedure, the reason for the description being fairly obvious. The use of such procedures is normally to be avoided - they tend to be highly inefficient - but occasionally the nature of a problem is such that it lends itself to solution by recursion. Note that in writing a recursive procedure in Elliott 4100 Algol all real, integer, Boolean and label parameters must be called by value.

Example 2 Write a recursive procedure FIB(n) for computing the nth. Fibonacci number, n≥0. The Fibonacci numbers f_0, f_1, f_2,---------- are defined by

$$f_0 = 1$$
$$f_1 = 1$$
$$f_n = f_{n-1} + f_{n-2},\ n = 2,3,4,5,\text{----------}$$

```
integer procedure FIB(n);
value n;
integer n;
FIB:= if n×(n-1)=0 then 1 else FIB(n-1) + FIB(n-2);
```

Example 3 Rewrite the function procedure FIB(n) to make it non-recursive.

```
integer procedure FIB(n);
value n;
integer n;
begin
      integer f0,f1,f2,j;
      f0:=f1:=1;
      for j:=2 step 1 until n do
      begin
            f2:=f0+f1;
            f0:=f1;
            f1:=f2
      end;
      FIB:=f1
end FIB;
```

Example 4 The following recursive procedure evaluates C(n,r) (i.e. ${}^{n}C_{r}$ or $\binom{n}{r}$), where $n \geq r \geq 0$, by using the well known result

$$C(n,r) = C(n-1,r) + C(n-1,r-1)$$

```
integer procedure C(n,r);
value n,r;
integer n,r;
C:= if (n-r)×r = 0 then 1 else C(n-1,r) + C(n-1,r-1);
```

Example 5 Rewrite the function procedure C(n,r) to make it non-recursive.

```
integer procedure C(n,r);
value n,r;
integer n,r;
begin
      integer j,t;
      t:=1;
      if n-r < r then r:=n-r;
      for j:=1 step 1 until r do t:=t×(n-j+1)÷j;
      C:=t
end;
```

16.6 Side-Effects

Consider the following program:-

```
begin
      real x,y,z,w,----------;
      integer i,j,p,----------;
      real procedure Sneaky(t);
      value t;
      real t;
      begin
            w:= w+1;
            Sneaky:= t+1/t
      end;
        ¦
        ¦
        ¦
        ¦
        ¦
        ¦
        ¦
```

Then whenever the procedure Sneaky is called into action the global variable w is increased by unity! This means, for example, that the instruction

```
y:= w × Sneaky(x)
```

has a different effect from that of the instruction

```
y:= Sneaky (x) × w
```

(why?). Procedures such as Sneaky, which alter the values of global variables when called into action, should normally be avoided. It is only too easy to forget about (or indeed to be unaware of) the side-effects of this type of procedure with disastrous consequences.

APPENDIX 1

ALGOL 60 BASIC SYMBOLS

An Algol 60 basic symbol is either a letter, a digit, a logical value or a delimiter.

(a) Letters

The letters in Algol 60 are

a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z|

A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z|

(b) Digits

The digits in Algol 60 are

0|1|2|3|4|5|6|7|8|9

(c) Logical Values

true | false

(d) Delimiters

The delimiters in Algol 60 are either operators, separators, brackets, declarators or specificators.

(i) Operators

+|-|×|/|÷|↑|

<|≤|=|>|≥|≠|

≡|⊃|∨|∧|⊣|

goto|if|then|else|for|do

(ii) Separators

, | . | $_{10}$ | : | ; | := | ␣ | step | until | while | comment

(iii) Brackets

| (|) | [|] | ' | ` | begin | end

(iv) Declarators

own | Boolean | integer | real | array | switch | procedure

(v) Specificators

string | label | value

APPENDIX 2

LIST OF REFERENCES

Textbooks

Dijkstra	: A Primer of Algol 60 Programming	(Academic Press)
Froberg	: Introduction to Algol Programming	(Oxford)
McCracken	: A Guide to Algol Programming	(Wiley)
Nicol	: Elementary Programming and Algol	(McGraw-Hill)
Rutishauser	: Description of Algol 60	(Springer-Verlag)
Wooldridge & Ratcliffe	: An Introduction to Algol Programming	(E.U.P.)

Computer Unit Documents

A Plain Man's Guide to Elliott 4130 Algol

Magnetic Tape Facilities in Elliott 4130 Algol

The Revised Report

Naur (Ed.): Revised Report on the Algorithmic Language Algol 60 appears in

(i) Communications of Association for Computing Machinery, Vol. 6 (1963), p. 1

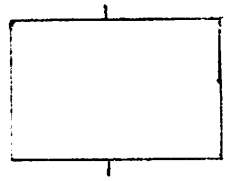
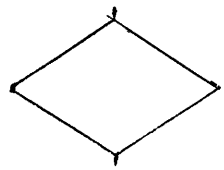
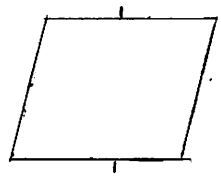
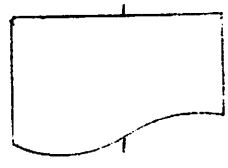
(ii) The Computer Journal, Vol. 5 (1963), p. 349

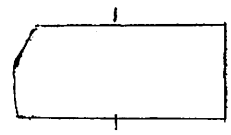
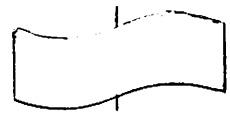
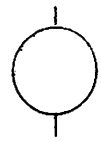
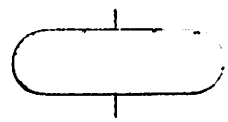
(iii) Numerische Mathematik, Vol. 4 (1963), p. 420

APPENDIX 3

Flow Chart Practice

The British Standard BS 4058: Part 1: 1966 recommends 30 flow chart symbols, the use of which will add greatly to the clarity of any flow chart. We list here several of the more important symbols and give an illustrative example:

Symbol number	Name and Definition	Symbol
1	Process Any kind of processing functions, e.g. process of executing a defined operation or group of operations resulting in a change of value, form, or location of information, or in the determination of which of several flow directions is to be followed.	
2	Decision A decision or switching type operation that determines which of a number of alternative paths is to be followed.	
12	Input/Output (I/O) Making available information for processing (input), or the recording of processed information (output).	
15	Document An I/O function in which the medium is a document	

Symbol Number	Name and Definition	Symbol
16	Punched Card An I/O function in which the medium is punched cards, including mark sense cards, partial cards, stub cards, mark scan cards, etc.	
19	Punched Tape An I/O function in which the medium is punched tape.	
28	Connector An exit to, or an entry from another part of the flowchart.	
29	Terminal, interrupt A terminal point in a flowchart; e.g. start, halt, delay or interrupt.	

Example 1 Redraw, using the British Standard flow chart symbols, the flow chart on page 21.